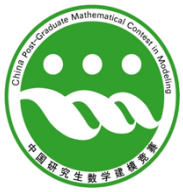


中国研究生创新实践系列大赛

“华为杯”第十七届中国研究生

数学建模竞赛

学　　校	南 京 大 学
参赛队号	20102840100
队员姓名	1. 胡鑫雯
	2. 乐盛捷
	3. 张贝贝

中国研究生创新实践系列大赛

“华为杯”第十七届中国研究生

数学建模竞赛

题 目　　F　飞行器质心平衡供油策略优化

摘　　要：

本文主要探讨了飞行器质心平衡供油策略，下面针对每个问题给出方法与结果。

针对问题一：在该任务中，主要探讨了飞行器发生俯仰时对质心的影响。由质心计算公式，主要要确定各油箱油的质心坐标。为了简化计算的复杂度，我们建立以油箱左下角为原点的直角坐标系，并将三维空间问题转化到二维平面讨论。分析可得，随着飞行器俯仰角度的变化，油面会有不同程度的倾斜，产生5种不同的形状，包括：矩形、三角形、2种梯形和五边形。通过对俯仰角θ的讨论，找到油面在不同形状间切换的临界值θ。根据飞行器俯仰角θ的正负值和油箱中的油量，分类讨论，求得在不同θ区间下的9个质心公式，从而计算出飞行器的质心坐标。结果中，飞行器的质心偏离原点的最大距离为1.1091米。本模型的算法复杂度为$O(n)$。

根据以上质心公式，可绘制飞行器在任务执行过程中质心变化曲线。观察发现，问题一在$t=1$时刻的质心与问题二附件3的第一个质心坐标相同。同时，依据质心变化曲线结果的连续性，验证了该部分模型求解的正确性与有效性。

针对问题二：在该任务中，主要探讨了不同供油策略对质心位置每时刻的影响。通过分析可知问题二是典型的单目标二次混合优化问题，在飞行器“油箱状态”、“同时供油”、“供油时间”、“供油速度”四个角度所提出的7个约束条件下，达到飞行器每一时刻质心位置$\vec{c_1}(t)$与理想质心位置$\vec{c_2}(t)$的欧氏距离的最大值达到最小的目标。通过枚举法，可列举34种满足约束条件的供油组合，在7200秒内则有34^{7200}种情况。由于问题的计算规模是指数级别，因此设计贪心策略，提出供油策略单目标优化模型。结合本文提出的“供油时长”、“供油速度”、“供油比例”和“供油总量”4条假设来简化计算，得到局部最优解，组成全局近似最优解。本模型的算法复杂度为$O(n)$。

计算得到，飞行器瞬时质心与理想质心距离的最大值为0.089183米，4个主油箱的总供油量为6441.52421千克，与发动机总耗油量一致。也绘制了6个油箱的供油速度曲线和4个主油箱的总供油速度曲线。算法中也兼顾了供油速度的平滑性。在验证阶段，可证明本文模型具有很好的有效性。

针对问题三：在该任务中，主要探讨了不同油箱初始载油量和供油策略对质心位置每时刻的影响。分析发现，问题三与问题二相比，多了要确定初始油量的任务。问题求解时，分为“初始油量分配”和“供油策略搜索”两个阶段。本文采样了5个初始总油量，通过两个单目标优化模型，得到质心间的欧式距离最大值与油量之间的关系，并进行拟合，发现当初始总油量为8080.1467千克时，得到最佳分配方案，此时欧式距离的最小值为0.172049米。在初始油量分配阶段，以最小化0时刻的飞行器质心与理想质心的距离为目标，在多种约束下建立油量分配单目标优化模型，通过差分优化算法来求得初始油量分配的近似最优解。在供油策略搜索阶段，可使用问题二中提出的供油策略单目标优化模型进行求解。本模型的算法复杂度为$O(n)$。

计算得到，1～6号油箱的最佳油量分配方案为[300，1645，1613.02，1684.08，2434.66，403.41]，飞行器质心与理想质心距离的最大值为0.172049米，4个主油箱的总供油量为6805.17467千克。也绘制了6个油箱的供油速度曲线和4个主油箱的总供油速度曲线。算法中也兼顾了供油速度的平滑性。在验证阶段，可证明本文模型具有很好的有效性。

针对问题四：在该任务中，主要探讨了不同俯仰角度和供油速度对质心位置每时刻的影响。分析发现，问题四与问题二有两方面的不同。其一是计算欧式距离的不同：由问题二计算与理想质心坐标的距离，变为计算与飞行器不载油时质心$\vec{c_0} = (0,0,0)$之间的距离；其二是有无俯仰角的不同：问题二无俯仰角，问题四有俯仰角，此区别可由问题一中求解得到的9种质心公式解决。因此问题四可以通过综合使用问题一质心公式与问题二的供油策略单目标优化模型求解。本模型的算法复杂度为$O(n)$。

计算得到，飞行器瞬时质心与飞行器不载油质心$\vec{c_0}$的最大距离达到最小时的值为0.241299米，4个主油箱的总供油量为7035.54516千克。也绘制了6个油箱的供油速度曲线和4个主油箱的总供油速度曲线。算法中也兼顾了供油速度的平滑性。在验证阶段，可证明本文模型也具有很好的有效性。

关键字：质心求解；单目标优化；差分优化；供油策略

目　录

一、问题重述

1.1 问题的背景

在飞行过程中，某一类飞行器有多个油箱，通过若干个油箱的联合供油以满足飞行任务要求和发动机的工作需求。在任务执行过程中，飞行器的质心[3]变化对飞行器的控制有着重要的影响，各个油箱内油量的分布和供油策略将导致飞行器质心的变化，进而影响到对飞行器的控制。因此，制定各个油箱的供油策略是这类飞行器控制的一项重要任务。油箱的供油策略可用其向发动机或其它油箱供油的速度曲线来描述。

1.2 问题的提出

针对问题一：已知某次任务中飞行器的 6 个油箱的供油速度及飞行器在飞行过程中的俯仰角变化数据，每秒记录一组数据。需要求出该飞行器在此次任务执行过程中的质心变化曲线，并将其质心按照时间先后顺序记录在结果表中。

针对问题二：已知某次任务的飞行器计划耗油速度数据，与飞行器在飞行器坐标系下的理想质心位置数据。根据任务需求，在飞行器始终保持平飞(俯仰角为 0)的任务规划过程中，为飞行器制定该次任务满足所有假设条件的 6 个油箱供油策略，使得飞行器每一时刻的质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小, i.e.

$$\min \max_t ||\vec{c}_1(t) - \vec{c}_2(t)||_2$$

针对问题三：假定初始油量未定，已知飞行器其他相关参数、某次任务的飞行器计划耗油速度数据，与飞行器在飞行器坐标系下的理想质心位置数据。在飞行器始终保持平飞(俯仰角为 0)的任务规划过程中，要为飞行器制定该次任务满足约束条件的 6 个油箱初始载油量及供油策略，使得本次任务结束时 6 个油箱剩余燃油总量至少 $1m^3$，并且飞行器每一时刻的质心位置$\vec{c}_1(t)$与理想质心位置$\vec{c}_2(t)$的欧氏距离的最大值达到最小, i.e,

$$\min \max_t ||\vec{c}_1(t) - \vec{c}_2(t)||_2。$$

针对问题四：在实际任务规划过程中，飞行器俯仰角随时间变化。已知飞行器俯仰角的变化数据和耗油速度数据。要为任务制定油箱供油策略，使得飞行器瞬时质心$\vec{c}_1(t)$与飞行器(不载油)质心$\vec{c}_0$的最大距离达到最小，i.e.

$$\min \max_t ||\vec{c}_1(t) - \vec{c}_0||_2。$$

二、问题分析及建模准备

2.1 问题一的分析

已知飞行器中的 6 个油箱的供油速度以及飞行器在飞行过程中的俯仰角变化数据，在飞行器保持平直飞行的条件下，要求计算飞行器的质心变化曲线，并将质心在飞行器坐标系下的位置数据按时间（每秒一组）先后顺序保存。

通过对问题的分析，可知这是一个典型的质心计算问题。先对不同的油量及俯仰角进行分类，然后分析每种情况下油箱质心的计算公式，最后带入数据得到油箱质心的具体位置。

2.2 问题二的分析

已知某次任务的飞行器计划耗油速度数据和飞行器在飞行器坐标系下的理想质心位置数据。在飞行器保持平飞的任务规划中，要求设计一套供油策略，使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。

通过对问题的分析，可知这是一个典型的单目标优化问题。在建立数学模型之前，可以通过梳理飞行器在飞行过程中所受的约束条件和目标函数，以确定飞行器在平飞情况下供油策略的单目标优化模型。

2.3 问题三的分析

已知某次任务的飞行器计划耗油速度数据和飞行器在飞行器坐标系下的理想质心位置数据。在飞行器始终保持平飞的任务规划中，问题三要求制定满足基本假设条件的 6 个油箱初始载油量及供油策略，目标是使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。初步分析，该问题仍然是一个目标优化问题。

2.4 问题四的分析

在实际任务规划过程中，飞行器俯仰角随时间变化而变化。已知飞行器俯仰角的变化数据与耗油速度数据，问题四要求制定油箱供油策略，目标仍是使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。

三、基本假设

飞行器的结构（如油箱的位置、形状、尺寸、供油关系、供油速度限制等）影响着油箱的供油策略和飞行器的质心变化。为简化问题，如图 1 所示，对飞行器的结构和相关供油限制作出以下假设和要求：

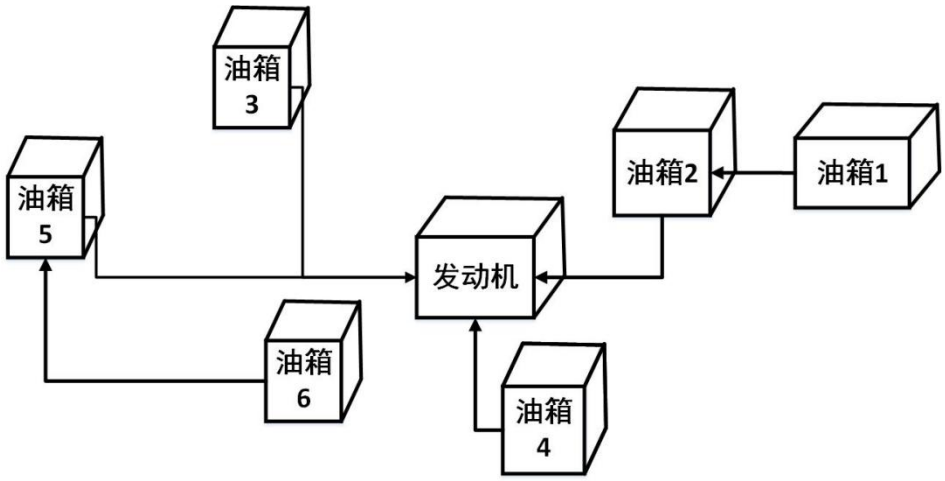


图 1：飞行器油箱供油示意图

（1）油箱均为长方体且固定在飞行器内部，设第i个油箱内部长、宽、高分别为a_i,b_i和c_i，$i = 1,2,\ldots,6$。长、宽、高的三个方向与飞行器坐标系的x, y, z轴三个方向平行。

（2）在飞行器坐标系下，飞行器（不载油）质心为$\overrightarrow{c_0}$（0，0，0），第i个空油箱中心位置为$\overrightarrow{\mathrm{P}_i}$，$i = 1,2,\ldots,6$。飞行器（不载油）总重量为 M。

（3）第i个油箱的供油速度上限为U_i (U_i>0) ，$i = 1,2,\ldots,6$。每个油箱一次供油的持续时间不少于 60 秒。

（4）主油箱 2、3、4、5 可直接向发动机供油，油箱 1 和油箱 6 作为备份油箱分别为油箱 2 和油箱 5 供油，不能直接向发动机供油。

（5）由于受到飞行器结构的限制，至多 2 个油箱可同时向发动机供油，至多 3 个油箱可同时供油。

（6）飞行器在执行任务过程中，各油箱联合供油的总量应至少满足发动机的对耗油量的需要（若某时刻供油量大于计划耗油量，多余的燃油可通过其它装置排出飞行器）。

（7）飞行器在飞行过程中可能发生姿态改变，主要是飞行航向上的上下俯仰或左右偏转。为简化问题，假设本文中飞行器姿态的改变仅考虑平直飞与俯仰情况。飞行器的俯仰将导致各油箱相对地面的姿态发生倾斜，在重力作用下，油箱的燃油分布也随之发生变化，从而使得飞行器质心发生偏移。

（8）俯仰角绝对值小于 90 度。

四、符号说明

符号	解释说明
m_0	飞行器的质量。
M_0	油箱中的初始总油量的质量。
M_k	第k个油箱的质量。
ρ	油的密度$850kg/m^3$
x_k	第k个油箱在x轴方向的中心位置
y_k	第k个油箱在y轴方向的中心位置
z_k	第k个油箱在z轴方向的中心位置
a_k	第k个油箱的长。
b_k	第k个油箱的宽。
c_k	第k个油箱的高。
$\overrightarrow{c_k}$	第k个油箱的质心坐标向量。
$\overrightarrow{c_1}(t)$	时刻t下本文所求得的飞行器质心位置。
$\overrightarrow{c_2}(t)$	时刻t下飞行器的理想质心位置。
$\overrightarrow{Q_k}(t)$	时刻t下本文所求得的第k个油箱的质心位置
$O_k(t)$	时刻t下第k个油箱的开关状态。
$M_k(t)$	时刻t下第k个油箱剩余油的质量。
$V_k(t)$	时刻t下第k个油箱剩余油的体积
$h_k(t)$	时刻t下第k个油箱剩余油的无俯仰角时的高度。
$v_k(t)$	时刻t下第k个油箱供油的速度。
$v_{engine}(t)$	时刻t下飞行器发送机所需的耗油速度
$v_{max}(k)$	第k个油箱参数中所规定的输送油量上限。
$S_k(t)$	时刻t下第k个油箱的油在xz轴二维平面构成的面积。
$Need(t)$	时刻t下飞行器所需要消耗油的质量。
$D(t)$	时刻t下飞行器质心欧氏距离目标函数
θ	飞行器俯仰角

五、问题一的求解

5.1 问题分析

已知飞行器中的 6 个油箱的供油速度以及飞行器在飞行过程中的俯仰角变化数据，在飞行器保持平直飞行的条件下，要求计算飞行器的质心变化曲线，并将质心在飞行器坐标系下的位置数据按时间（每秒一组）先后顺序保存。

通过对问题的分析，可知这是一个典型的**质心计算问题**。先对不同的油量及俯仰角进行分类，然后分析每种情况下油箱质心的计算公式，最后带入数据得到油箱质心的具体位置。

质心公式如下所示：

$$\overrightarrow{c_1}(t) = \frac{m_0 * \vec{0} + \sum_{k=1}^{6} M_k(t)\,\overrightarrow{Q_k}(t)}{m_0 + \sum_{k=1}^{6} M_k(t)} \tag{1}$$

5.2 模型建立

5.2.1 模型预处理

（1）“三维空间转二维平面”：通过对问题一进行分析，可以发现当飞行器平直飞行，或俯仰角发生变化时，油箱中的油在y轴方向并不会发生改变。如果将油箱中的油看作一个立方体，宽是固定的，因此可以将一个三维空间转换为二维平面进行讨论。

（2）“油箱侧面变化情况”： 分析可得，随着飞行器俯仰，油面会有不同程度的倾斜，产生 5 种不同的形状，包括矩形、三角形、梯形（2 种）和五边形。

倾斜角度与飞行器俯仰角度一致，而根据飞行器的俯仰角正负情况以及油的含量，5 种形状可分析得 9 个质心公式，具体可看 5.2.3 油面质心求解。

（3）“油箱左下角为分析原点”：为了方便求解，问题一的质心计算都先以油箱左下角A点暂时作为分析的原点，已知第k个油箱在x、y、z轴方向的中心位置分别表示为x_k、y_k、z_k，油箱的长宽高分别为a_k、b_k、c_k，通过以下公式（2）～（4），可得每个油箱A点的坐标为（x_{k_A},y_{k_A},z_{k_A}）。

$$x_{k_A} = x_k - a_k * \frac{1}{2} \tag{2}$$

$$y_{k_A} = y_k - b_k * \frac{1}{2} \tag{3}$$

$$z_{k_A} = z_k - c_k * \frac{1}{2} \tag{4}$$

计算出以A点为原点的油箱的质心$\overrightarrow{Q_k'}(t)(x', y', z')$后，再加上$A$点的坐标即可得到油体相对于飞行器的质心坐标：

$$\overrightarrow{Q_k}(t) = (x_{k_A} + x', y_{k_A} + y', z_{k_A} + z')$$

（4）“油箱质量的求取”： 根据油箱每秒的的供油速度$v_k(t)$，可以得到t时刻时油箱中的油的质量。根据题目油箱的质量分为两种情况讨论：情况一，油箱质量减小，可得到公式（5）；情况二，油箱质量可能增加，可得到公式（6），如油箱 2，5。

$$M_k(t) = M_k(t-1) - v_k(t) * 1, \quad k = 1,2,3,4,5,6 \tag{5}$$

$$M_k(t) = M_k(t-1) + v_k(t) * 1, \quad k = 2,5 \tag{6}$$

从飞行器参数文件中得知每个油箱中油的初始体积$V_k(0)$和密度$\rho = 850kg/m^3$，可以得到油的初始质量$M_k(0)$如下：

$$M_k(0) = V_k(0) * \rho \tag{7}$$

5.2.2 临界值θ的讨论

当$\theta = 0$，飞行器水平前行，油面呈现为 “矩形”；当$\theta \neq 0$，油面会呈现多种形状，在不同形状下，质心公式的求解不同。因此讨论在不同形状间切换的临界值θ，是非常有必要的。

（1）油的体积小于或等于油箱体积的一半时，即$h_k(t) \leq \frac{c_k}{2}$，油面只会呈现梯形或三角形形状。

情况一：θ较小的时候，飞行器只会轻微倾斜，此时油面呈梯形形状。当倾斜角度θ增大到θ_1，油面之后会变成三角形形状，如图 2 所示。

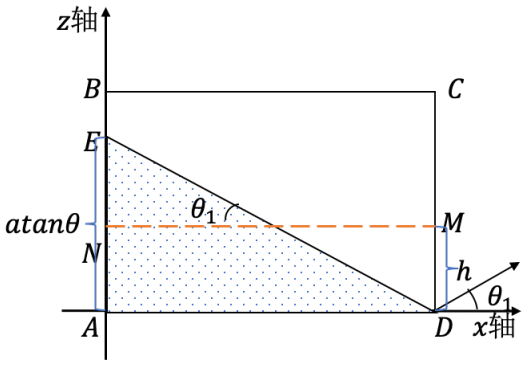


图 2：油面形状由梯形转入三角形临界点示意图

由于油体体积一定，y 轴方向没有运动，即油体的长不变，故可以推断油体的侧面积即使倾斜也不会变，即不管油面是矩形还是其他形状，面积都是相等的。设$S_k(t)$为时刻t下第k个油箱的油在xz轴二维平面构成的面积，则：

$$S_k(t)=\frac{V_k(t)}{b_k}=\frac{\frac{M_k(t)}{\rho}}{b_k} \tag{8}$$

又因为：

$$S_{EAD}=S_k(t)=\frac{AE*AD}{2}=\frac{a_k\tan\theta_1*a_k}{2} \tag{9}$$

其中S_{EAD}代表三角形 EAD 的面积，根据公式(8)计算出的$S_k(t)$可以解得：

$$\theta_1=\tan^{-1}\frac{2M_k(t)}{\rho b_k a_k^2} \tag{10}$$

情况二：倾斜角度θ继续增大，直到增大到θ_2，之后油面将变成梯形形状，如图 3 所示。

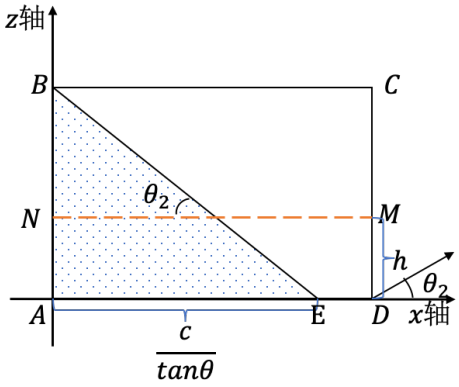


图 3：油面形状由三角形转入梯形临界点示意图

由于侧面积$S_k(t)$不变，则：

$$S_k(t)=S_{BAE}=\frac{AB*AE}{2}=\frac{c_k*\frac{c_k}{\tan\theta_2}}{2} \tag{11}$$

其中S_{BAE}代表三角形 BAE 的面积，根据公式(8)计算出的$S_k(t)$则可以解得：

$$\theta_2 = \tan^{-1}\frac{\rho b_k c_k^2}{2M_k(t)} \tag{12}$$

（2）油的体积大于油箱体积的一半时,即 $h_k(t) > \frac{c_k}{2}$，油面只会呈现梯形或五边形形状。

情况三：当倾斜角度θ较小的时候，飞行器只会轻微倾斜，此时油面呈梯形形状。当倾斜角度θ增大到θ_3，油面之后变成五边形形状，如图 4 所示。

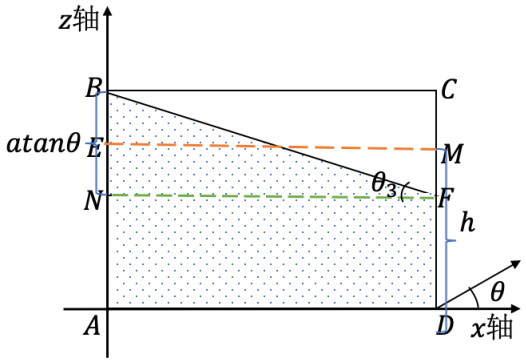


图 4：油面形状由梯形转入五边形临界点示意图

由于侧面积$S_k(t)$不变，可得：

$$S_k(t) = S_{BNF} + S_{ANFD} = \frac{BN * NF}{2} + AD * AN = \frac{a_k tan\theta_3 a_k}{2} + a_k(c_k - a_k tan\theta_3)$$

其中S_{BNF}, S_{ANFD}分别代表三角形 BNF 和矩形 ANFD 的面积，根据公式(8)计算出的$S_k(t)$则可以解得：

$$\theta_3 = \tan^{-1}\frac{2\left(c_k a_k - \frac{M_k(t)}{\rho b_k}\right)}{a_k^2} \tag{13}$$

情况四：倾斜角度θ继续增大，直到θ_4，之后油面变回**梯形**形状，如图 5 所示。

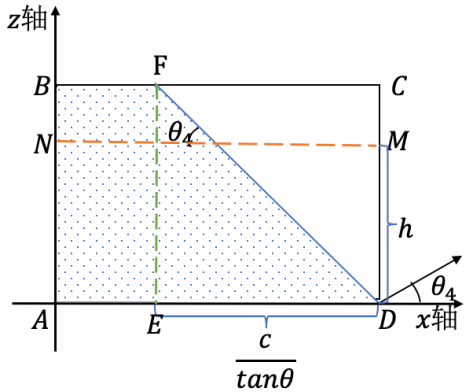


图 5：油面形状由五边形转入梯形临界点示意图

由侧面积不变可得：

$$S_k(t) = S_{FED} + S_{ABFE} = \frac{FE * ED}{2} + AB * AE = \frac{c_k * \frac{c_k}{tan\theta_4}}{2} + c_k\left(a_k - \frac{c_k}{tan\theta_4}\right) \tag{14}$$

其中S_{FED}, S_{ABFE}分别代表三角形 FED 和矩形 ABFE 的面积，根据公式(8)计算出的$S_k(t)$则可以解得：

$$\theta_4 = \tan^{-1}\frac{c_k^2}{2\left(c_k a_k - \frac{M_k(t)}{\rho b_k}\right)} \tag{15}$$

5.2.3 油面质心求解

（1）油面呈现为“矩形”： $\theta = 0$，如图 6 所示。

如果俯仰角θ=0，代表飞行器水平前行，油面将呈现**矩形**形状。

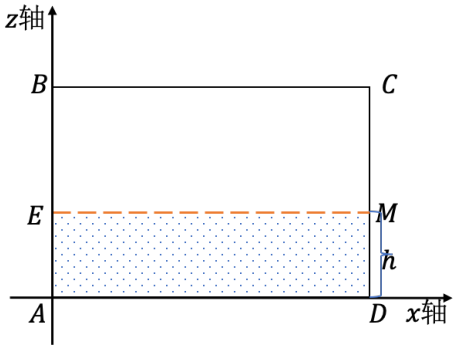


图 6：油面形状为矩形示意图

$h_k(t)$代表第k个油箱在飞行器平飞时，油箱形成矩形油面的高，已知$S_k(t)$为时刻t下第k个油箱的油在xz轴二维平面构成的面积，所以可由下式计算所得：

$$h_k(t) = \frac{S_k(t)}{AD} = \frac{S_k(t)}{a_k} = \frac{V_k(t)}{a_k b_k} = \frac{M_k(t)}{\rho a_k b_k} \tag{16}$$

此时油体的质心为：

$$\overrightarrow{Q_k}'(t) = \left(\frac{AD}{2}, \frac{b_k}{2}, \frac{EA}{2}\right) = \left(\frac{a_k}{2}, \frac{b_k}{2}, \frac{h_k(t)}{2}\right) \tag{17}$$

（2）油面呈现为“三角形”： $h_k(t) \le \frac{c_k}{2}$并且$\theta_1 \le |\theta|$并且$|\theta| \le \theta_2$，如图 7 所示。

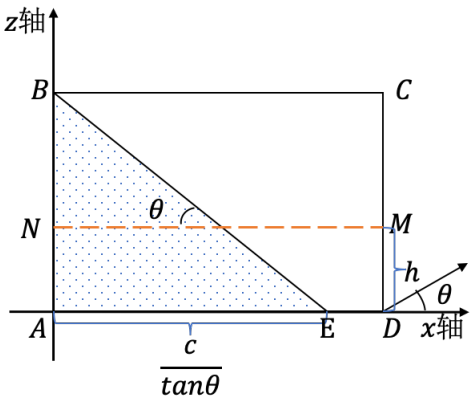


图 7：油面形状为三角形示意图

如果θ为正：

已知$S_k(t)$为时刻t下第k个油箱的油在xz轴二维平面构成的面积，假设$AF=x$可得：

$$S_k(t)=S_{EAF}=\frac{AE*AF}{2}=\frac{xtan\theta*x}{2} \tag{18}$$

其中S_{EAF}代表三角形 EAF 的面积，结合公式(8)可得：

$$x=\sqrt{\frac{2M_k(t)}{\rho b_k tan\theta}} \tag{19}$$

此时油箱的质心为：

$$\overrightarrow{Q_k}'(t)=\left(\frac{x_E+x_A+x_F}{3},\frac{b_k}{2},\frac{y_E+y_A+y_F}{3}\right) \tag{20}$$
$$=\left(\frac{0+0+AF}{3},\frac{b_k}{2},\frac{AE+0+0}{3}\right)=(\frac{x}{3},\frac{b_k}{2},\frac{xtan\theta}{3})$$

如果θ为负：计算过程类似于θ为正的步骤，最后得到：

$$\overrightarrow{Q_k}'(t)=\left(b_k-\frac{x}{3},\frac{b_k}{2},\frac{xtan\theta}{3}\right) \tag{21}$$

（3）油面呈现为“梯形 1”：$|\theta|<\theta_1$且$h_k(t)\le\frac{c_k}{2}$；$|\theta|<\theta_3$且$h_k(t)>\frac{c_k}{2}$。如图 8 所示。

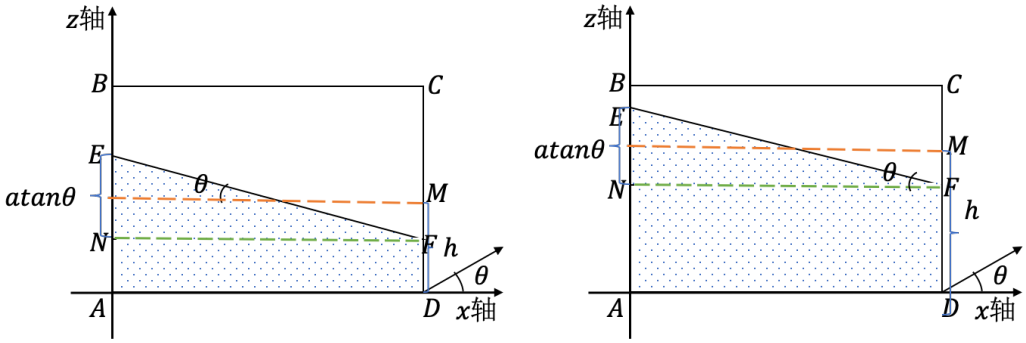


图 8：$h_k(t)\le\frac{c_k}{2}$和$h_k(t)>\frac{c_k}{2}$时油面形状为梯形 1 示意图

如果θ为正：

已知$S_k(t)$为时刻t下第k个油箱的油在xz轴二维平面构成的面积，假设$AN=x$可得：

$$S_k(t)=S_{ENF}+S_{ANFD}=\frac{NE*NF}{2}+NA*AD=\frac{a_k tan\theta*a_k}{2}+xa_k \tag{22}$$

其中S_{ENF}和S_{ANFD}分别代表三角形 ENF 和矩形 NADF 的面积，结合公式(8)可得：

$$x=\frac{M_k(t)}{a_k b_k\rho}-\frac{a_k tan\theta}{2} \tag{23}$$

此时油箱的质心为：

$$\overrightarrow{Q_k}'(t)=\frac{M_{k_{ENF}}(t)*\overrightarrow{Q_{k_{ENF}}}(t)+M_{k_{NADF}}(t)*\overrightarrow{Q_{k_{NADF}}}(t)}{M_{k_{ENF}}(t)+M_{k_{NADF}}(t)}$$

$$=\frac{S_{ENF}b_k\rho\left(\frac{x_E+x_N+x_F}{3},\frac{b_k}{2},\frac{y_E+y_N+y_F}{3}\right)+S_{ANFD}b_k\rho(\frac{x_D}{2},\frac{b_k}{2},\frac{y_N}{2})}{S_{ENF}b_k\rho+S_{ANFD}b_k\rho}$$

$$=\frac{\frac{a_k tan\theta*a_k}{2}\left(\frac{0+0+a_k}{3},\frac{b_k}{2},\frac{0+0+x}{3}\right)+xa_k(\frac{a_k}{2},\frac{b_k}{2},\frac{x}{2})}{S_k(t)}$$

$$=\frac{\frac{a_k tan\theta*a_k}{2}\left(\frac{a_k}{3},\frac{b_k}{2},\frac{x}{3}\right)+xa_k(\frac{a_k}{2},\frac{b_k}{2},\frac{x}{2})}{\frac{M_k(t)}{b_k\rho}} \tag{24}$$

其中$M_{k_{ENF}}(t)$和$M_{k_{NADF}}(t)$分别代表三角形 ENF 和矩形 NADF 对应的油体的质量，$\overrightarrow{Q_{k_{ENF}}}(t)$和$\overrightarrow{Q_{k_{NADF}}}(t)$分别代表三角形 ENF 和矩形 NADF 对应的油体的相对点 A 的质心位置。

如果θ为负：计算过程类似于θ为正的步骤，最后得到：

$$\overrightarrow{Q_k}'(t)=\frac{\frac{a_k tan\theta*a_k}{2}\left(\frac{2a_k}{3},\frac{b_k}{2},\frac{x}{3}\right)+xa_k\left(\frac{a_k}{2},\frac{b_k}{2},\frac{x}{2}\right)}{\frac{M_k(t)}{b_k\rho}} \tag{25}$$

（4）油面呈现为“梯形 2”：$|\theta|>\theta_2$且$h_k(t)\le\frac{c_k}{2}$;$|\theta|>\theta_4$且$h_k(t)>\frac{c_k}{2}$。如图 9 所示。

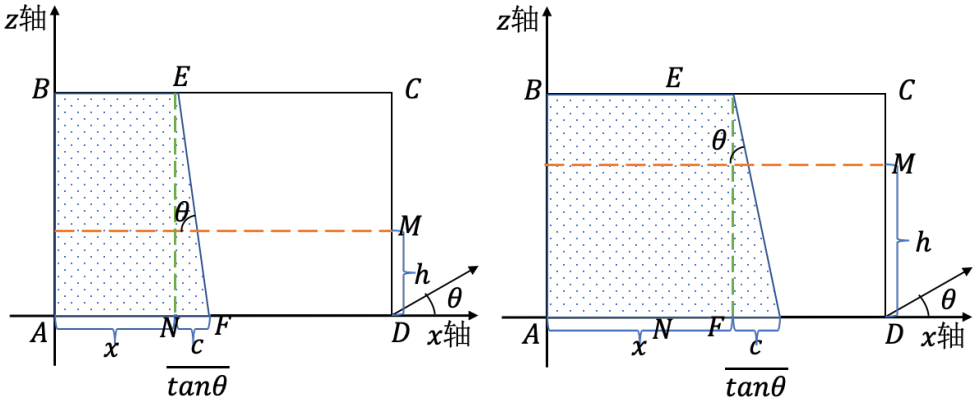


图 9：$h_k(t)\le\frac{c_k}{2}$和$h_k(t)>\frac{c_k}{2}$时油面形状为梯形 2 示意图

如果θ为正：

已知$S_k(t)$为时刻t下第k个油箱的油在xz轴二维平面构成的面积，假设$AN=x$可得：

$$S_k(t)=S_{ENF}+S_{ANEB}=\frac{NE*NF}{2}+NA*AB=\frac{c_k\frac{c_k}{tan\theta}}{2}+xc_k \tag{26}$$

其中S_{ENF}和S_{ANEB}分别代表三角形 ENF 和矩形 NADF 的面积，结合公式(8)可得：

$$x = \frac{M_k(t)}{c_k b_k \rho} - \frac{c_k}{2tan\theta} \tag{27}$$

此时油箱的质心为：

$$\begin{aligned}
\overrightarrow{Q_k}'(t) &= \frac{M_{k_{ENF}}(t) * \overrightarrow{Q_{k_{ENF}}}(t) + M_{k_{ANEB}}(t) * \overrightarrow{Q_{k_{ANEB}}}(t)}{M_{k_{ENF}}(t) + M_{k_{ANEB}}(t)} \\
&= \frac{S_{ENF} b_k \rho \left(\frac{x_E + x_N + x_F}{3}, \frac{b_k}{2}, \frac{y_E + y_N + y_F}{3}\right) + S_{ANEB} b_k \rho (\frac{x_N}{2}, \frac{b_k}{2}, \frac{y_B}{2})}{S_{ENF} b_k \rho + S_{ANEB} b_k \rho} \\
&= \frac{\frac{c_k \frac{c_k}{tan\theta}}{2} \left(\frac{x + x + x + \frac{c_k}{tan\theta}}{3}, \frac{b_k}{2}, \frac{c_k + 0 + 0}{3}\right) + x c_k (\frac{x}{2}, \frac{b_k}{2}, \frac{c_k}{2})}{S_k(t)} \\
&= \frac{\frac{c_k \frac{c_k}{tan\theta}}{2} \left(x + \frac{c_k}{3tan\theta}, \frac{b_k}{2}, \frac{c_k}{3}\right) + x c_k \left(\frac{x}{2}, \frac{b_k}{2}, \frac{c_k}{2}\right)}{\frac{M_k(t)}{b_k \rho}}
\end{aligned} \tag{28}$$

其中$M_{k_{ENF}}(t)$和$M_{k_{ANEB}}(t)$分别代表三角形 ENF 和矩形 ANEB 对应的油体的质量，$\overrightarrow{Q_{k_{ENF}}}(t)$和$\overrightarrow{Q_{k_{ANEB}}}(t)$分别代表三角形 ENF 和矩形 ANEB 对应的油体的相对点 A 的质心位置。

如果θ为负：计算过程类似于θ为正的步骤，最后得到：

$$\overrightarrow{Q_k}'(t) = \frac{\frac{c_k \frac{c_k}{tan\theta}}{2} \left(a_k - x - \frac{c_k}{3tan\theta}, \frac{b_k}{2}, \frac{c_k}{3}\right) + x c_k \left(a_k - \frac{x}{2}, \frac{b_k}{2}, \frac{c_k}{2}\right)}{\frac{M_k(t)}{b_k \rho}} \tag{29}$$

（5）油面呈现为“五边形”： $h_k(t) \le \frac{c_k}{2}$且$\theta_3 \le |\theta| \le \theta_4$，如图 10 所示。

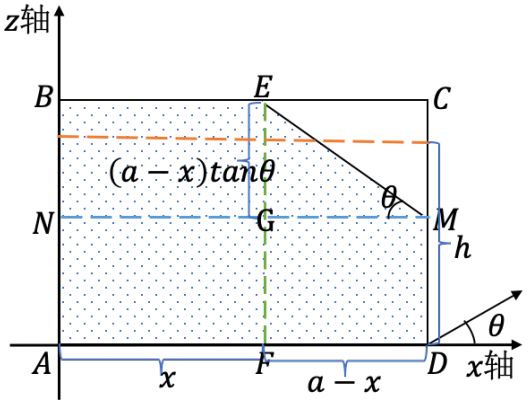


图 10：油面形状为五边形示意图

如果θ为正：

已知$S_k(t)$为时刻t下第k个油箱的油在xz轴二维平面构成的面积，假设$AF = x$可得：

$$S_k(t) = S_{EGM} + S_{ABEF} + S_{FGMD} = \frac{EG * GM}{2} + AF * AB + FD * FG$$
$$= \frac{(a_k - x)^2 tan\theta}{2} + xc_k + (a_k - x)(c_k - (a_k - x)tan\theta) \quad (30)$$

其中S_{EGM}和S_{ABEF}和S_{FGMD}分别代表三角形 EGM 和矩形 ABEF 和矩形 FGMD 的面积，结合公式(8)可得：

$$x = a_k - \sqrt{\frac{2\left(a_k c_k - \frac{M_k(t)}{b_k \rho}\right)}{tan\theta}} \quad (31)$$

此时油箱的质心为：

$$\overrightarrow{Q_k}'(t) = \frac{M_{k_{EGM}}(t) * \overrightarrow{Q_{k_{EGM}}}(t) + M_{k_{ABEF}}(t) * \overrightarrow{Q_{k_{ABEF}}}(t) + M_{k_{FGMD}}(t) * \overrightarrow{Q_{k_{FGMD}}}(t)}{M_{k_{EGM}}(t) + M_{k_{ABEF}}(t) + M_{k_{FGMD}}(t)}$$

$$= \frac{\frac{(a_k - x)^2 tan\theta}{2}\left(\frac{x + x + a_k}{3}, \frac{b_k}{2}, \frac{c_k + 2(c_k - (a_k - x)tan\theta)}{3}\right)}{S_k(t)}$$

$$+ \frac{xc_k\left(\frac{x}{2}, \frac{b_k}{2}, \frac{c_k}{2}\right) + (a_k - x)(c_k - (a_k - x)tan\theta)\left(\frac{a_k + x}{2}, \frac{b_k}{2}, \frac{(c_k - (a_k - x)tan\theta)}{2}\right)}{S_k(t)}$$

如果θ为负：计算过程类似于θ为正的步骤，最后得到：

$$\overrightarrow{Q_k}'(t)$$

$$= \frac{\frac{(a_k - x)^2 tan\theta}{2}\left(\frac{2a_k - 2x}{3}, \frac{b_k}{2}, \frac{c_k + 2(c_k - (a_k - x)tan\theta)}{3}\right)}{S_k(t)}$$

$$+ \frac{xc_k\left(a_k - \frac{x}{2}, \frac{b_k}{2}, \frac{c_k}{2}\right) + (a_k - x)(c_k - (a_k - x)tan\theta)\left(\frac{a_k - x}{2}, \frac{b_k}{2}, \frac{(c_k - (a_k - x)tan\theta)}{2}\right)}{S_k(t)}$$

5.3 模型求解

5.3.1 算法实现的关键步骤

表 1：基于质心公式的代码实现主要思路

基于质心公式的代码实现主要思路
（1）根据油箱的参数计算出各油箱油的初始质量、油箱左下角A点的位置。 （2）在 t 时刻，获得对应的供油速度和俯仰角，其中 1 号油箱或 6 号油箱进行供油时，二号油箱或五号油箱油量增加。 （3）对每一个油箱： (a) 根据供油速度计算出当前的油箱质量； (b) 根据当前质量，可以获得油体体积，从而得到矩形油面的高$h_k(t)$；

(c) 获得当前的俯仰角度θ，结合油箱参数与h，得到θ对应的域，选择相应的质心计算公式计算出油箱的相对质心；

(d) 将相对质心的位置加上A点坐标，获得油箱相对飞行器的质心；

（4）根据质心计算公式结合飞行器本身以及所有油箱的质心及质量，获得这一刻飞行器整体的质心位置。

5.3.2 算法实现伪代码

表 2：基于质心公式的代码实现伪代码

基于质心公式的代码实现伪代码

```
for box in box_list:
    calculate_boxAs
for t in time_interval:
    m_list=box_ms[t]
    v_list=box_vs[t]
    theta = theta_list[t]
    for m,v in m_list,v_list:
        calculate_current_m,calculate_current_v,calculate_current_V
        h=current_V/ ρ
        if h<=c/2:
            if theta<theta1: 使用梯形 1 公式计算 mass_center
            if theta1<=theta<=theta2: 使用三角形公式计算 mass_center
            if theta>theta2: 使用梯形 2 公式计算 mass_center
         if h>c/2:
            if theta<theta3: 使用梯形 1 公式计算 mass_center
            if theta3<=theta<=theta4: 使用五边形公式计算 mass_center
            if theta>theta4: 使用梯形 2 公式计算 mass_center
    calculate_planet_mass_center
```

5.4 模型结果

（1）关键质心数据：经过上述模型建立和模型求解，本文可求解得到随时间变化的质心结果。下面将对关键时刻的质心求解结果进行展示，如表 3 所示：

表 3：关键质心表

时刻	x	y	z	备注
1	1.257e-09	6.285e-10	3.567e-09	**第 1 秒，**没有任何油箱进行供油
2	-2.268e-07	4.661e-08	-1.356e-08	**俯仰角一直为 0，**有油箱开始供油
66	-0.002	9.8638e-05	-3.050e-05	**俯仰角开始变化，**并持续供油

7200	-0.058	-0.031	-0.106	**最后一秒**

（2）质心变化曲线：已知质心为三维空间的点，又因质心随着时间的变化而变化。因此，为展示质心变化曲线，需要四维立体空间。本文求解得到的质心变化曲线，如图 11 所示。其中横坐标为x轴，纵坐标为y轴，竖直方向为 z 轴，第四维使用颜色的深浅来表示时间t的推移。当$t = 1$时颜色最深（黑色），$t = 7200$时颜色最浅（白色）。

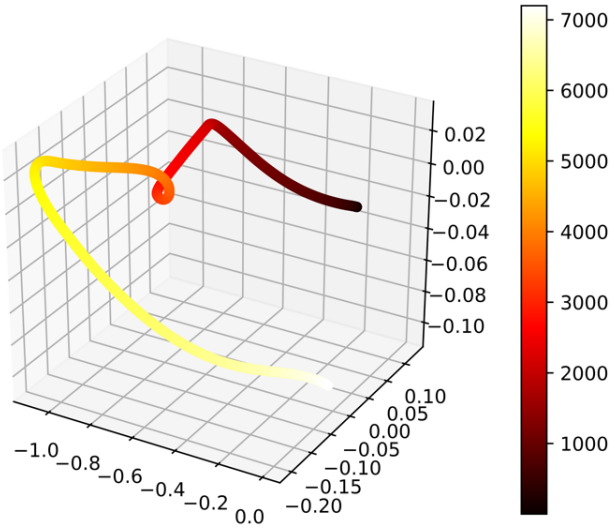


图 11：飞行器质心变化曲线图

5.5 模型有效性验证

首先，由于问题 2 提供的附件是没有任何供油和俯仰角下的理想质心情况，和本题t = 1时的情况相同，而在该时刻本文提供的结果和其数据相同，这可以在一定程度上验证本文的方法的正确性。

同时，通过观察供油速度和俯仰角数据，可以看到它们每秒的变化量都很小，并呈现一定的连续性，从而可以推断质心的分布必然也是近似连续的。本文提供的质心变化曲线图(图 11)的连续性也在这个意义上验证了我们方法的正确性。

本文还观察了每一个质心分量 x，y，z 随 t 的变化情况，如图 12 所示。可以看出 x，y，z 的变化都成连续趋势，很好的匹配了供油速度和俯仰角连续变化的特性。

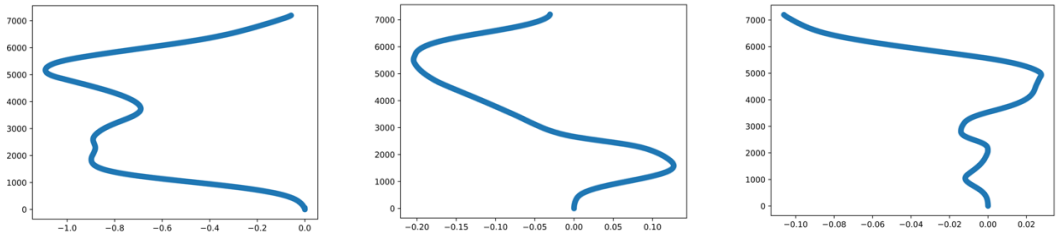


图 12：x, y, z轴数据分别随时间t变化曲线图

另外，本文补充了六个油箱单独的质心变化曲线，如图 13（a）~（f）。其连续性间接佐证了本文结果的合理性。

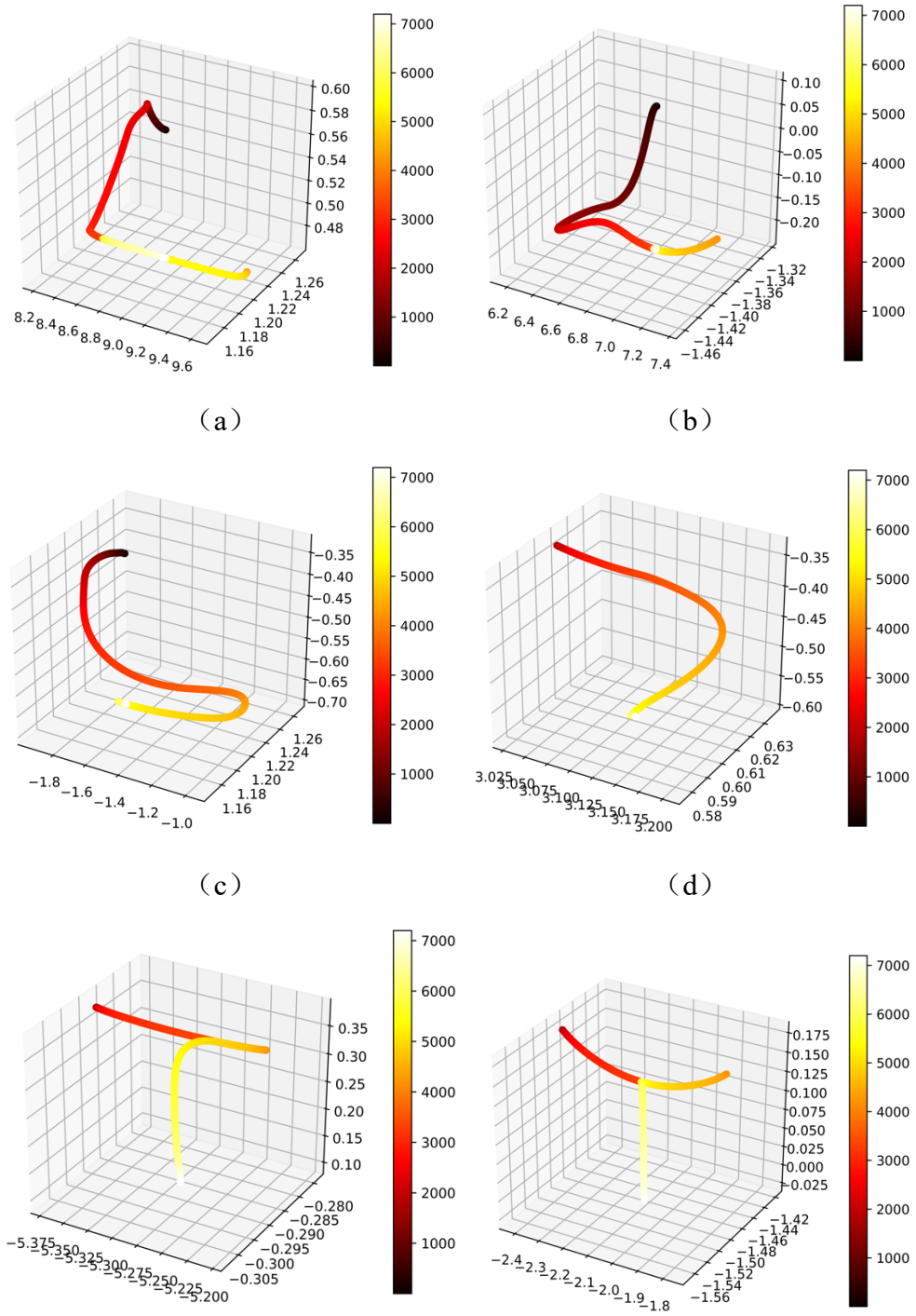


图 13：（a）~（f）分别对应 1～6 号的 6 个油箱各自的质心变化曲线

六、问题二的求解

6.1 问题分析

已知某次任务的飞行器计划耗油速度数据和飞行器在飞行器坐标系下的理想质心位置数据。在飞行器保持平飞的任务规划中，要求设计一套供油策略，使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。

通过对问题的分析，可知这是一个典型的单目标优化问题。在建立数学模型之前，通过梳理飞行器在飞行过程中所受的约束条件和目标函数，以确定飞行器在平飞情况下供油策略的单目标优化模型。

6.2 模型建立

6.2.1 约束条件

（1）“油箱状态”约束：开与关。本文用$O_k(t)$表示第k个油箱在时刻t下的开关状态，$O_k(t)$是一个示性函数，即：

$$O_k(t)=\begin{cases}1，k\text{油箱打开}\\0，k\text{油箱关闭}\end{cases} \tag{32}$$

（2）“同时供油”约束：至多 2 个油箱同时向发动机供油，至多 3 个油箱同时供油。主油箱 2、3、4、5 可直接向发动机供油，油箱 1 和油箱 6 作为备份油箱分别为油箱 2 和油箱 5 供油，不能直接向发动机供油。可将上述约束表达为：

$$1\le\sum_{k=2}^{5}O_k(t)\le 2 \tag{33}$$

$$2\le\sum_{k=1}^{6}O_k(t)\le 3，\ if\,O_1(t)+O_6(t)=2 \tag{34}$$

$$2\le\sum_{k=1}^{6}O_k(t)\le 3，\ if\,O_1(t)+O_6(t)=2 \tag{35}$$

（3）“供油时间”约束：每个油箱一次供油的持续时间不少于 60 秒。假设$T+1$为一个油箱最短的连续供油时间长度（单位为“秒”），该约束可以描述为：

$$if\,O_k(t_i)-O_k(t_{i-1})=1,\quad i=1,2,\dots,7200-T \tag{36}$$

$$Then,\quad\sum_{j=i}^{i+T}O_k(t_j)=T+1，\ T\ge 59 \tag{37}$$

（4）“供油速度”约束：四个主油箱的供油速度之和大于或等于飞行器所需耗油的速度（等价于四个主油箱某时刻的供油质量之和大于或等于飞行器所需耗油的质量）。同时要满足，每个油箱的供油速度不能超过该油箱参数所规定的输送油量上限。可表达为：

$$\sum_{k=2}^{5} O_k(t_i)(M_k(t_{i-1}) - M_k(t_i)) \geq Need(t_i), \quad i = 1,2,\dots,7200 \tag{38}$$

$$M_k(i) = M_k(0) - \sum_{j=1}^{i} v_k(j), \quad i = 1,2,\dots,7200 \tag{39}$$

$$v_k(t) \leq v_{max}(k) \tag{40}$$

上述公式中以$Need(t)$表示飞行器在t时刻下所需要消耗油的质量，$v_k(t)$表示第k个油箱在时刻t下的供油的速度（单位为“千克/秒”），$M_k(t)$表示第 k 个油箱在时刻t下的剩余油的质量，$v_{max}(k)$表示第k个油箱参数中所规定的输送油量上限。

6.2.2 目标函数

在飞行器保持平飞的任务规划中，目标是使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。可数学表达为：

$$D(t) = \min_t \max \| \overrightarrow{c_1}(t) - \overrightarrow{c_2}(t) \|_2 \tag{41}$$

$$\| \overrightarrow{c_1}(t) - \overrightarrow{c_2}(t) \|_2 = \sqrt{\left(x_1(t) - x_2(t)\right)^2 + \left(y_1(t) - y_2(t)\right)^2 + \left(z_1(t) - z_2(t)\right)^2}$$

$$s.t. \begin{cases} 1 \leq \sum_{k=2}^{5} O_k(t) \leq 2 \\ 2 \leq \sum_{k=1}^{6} O_k(t) \leq 3, \ if O_1(t) + O_6(t) = 2 \\ 2 \leq \sum_{k=1}^{6} O_k(t) \leq 3, \ if O_1(t) + O_6(t) = 2 \\ if O_k(t_i) - O_k(t_{i-1}) = 1, \\ \sum_{j=i}^{i+T} O_k(t_j) = T + 1, \ i = 1,2,\dots,7200 - T, T \geq 59 \\ \sum_{k=2}^{5} O_k(t_i)(M_k(t_{i-1}) - M_k(t_i)) \geq Need(t_i), \quad i = 1,2,\dots,7200 \\ M_k(i) = M_k(0) - \sum_{j=1}^{i} v_k(j), \quad i = 1,2,\dots,7200 \\ v_k(t) \leq v_{max}(k) \end{cases} \tag{42}$$

$$\overrightarrow{c_1}(t) = \{ x_1(t), \ y_1(t), z_1(t) \} \tag{43}$$

$$\overrightarrow{c_2}(t) = \{ x_2(t), \ y_2(t), z_2(t) \} \tag{44}$$

其中$x_1(t)$, $y_1(t)$, $z_1(t)$表示为本文所求得的质心位置的空间坐标，$x_2(t), y_2(t), z_2(t)$表示为理想质心位置的空间坐标。

6.2.3 单目标优化模型

对于$\overrightarrow{c_1}(t)$，已知飞行器在不载油时的质心坐标是$(0,0,0)$，载油时，各个油箱中的油会影响质心位置。假设$M_k(t)$表示第k个油箱在时刻t下的剩余油质量的函数，i为油箱 1～6 的编号。通过油箱的质量和油的密度 $850kg/m^3$，因此，通过公式 X，可得油箱中油的体积为：

$$V_k(t) = \frac{M_k(t)}{850} \tag{45}$$

因为题干中说明飞机没有俯仰角，又已知油箱的长宽高分别为a_k、b_k、c_k，因此体积除以底面积可得油箱中油的高度$h_k(t)$、油箱的质心$\overrightarrow{Q_k}(t)$，从而可推导出飞行器整体的质心位置$\overrightarrow{c_1}(t)$：

$$h_k(t) = \frac{M_k(t)}{850 * a_k * b_k} \tag{46}$$

$$\overrightarrow{Q_k}(t) = \left(x_k, y_k, z_k + \frac{h_k(t) - c_k}{2}\right) \tag{47}$$

$$\overrightarrow{c_1}(t) = \frac{m_0 * \vec{0} + \sum_{k=1}^{6} M_k(t)\,\overrightarrow{Q_k}(t)}{m_0 + \sum_{k=1}^{6} M_k(t)} \tag{48}$$

下面要计算$M_k(t)$ 和 $\overrightarrow{Q_k}(t)$。$\overrightarrow{Q_k}(t)$由$M_k(t)$决定，$M_k(t)$由供油策略决定，供油策略为每一时刻各个油箱的开关情况和输油速度（关闭的油箱输出速度为 0）。**因此，问题二的核心在于供油组合的选择和供油速度的取值。**

在模型求解阶段，本文将会依据上述约束条件和目标函数，设计一个尽可能好的供油策略，其中供油策略包括在不同时刻下，不同油箱的开关情况和输油速度。

6.3 模型求解

通过枚举法，可以列举出 34 种满足约束条件的供油组合。具体可看下表 4：

表 4：满足约束条件的供油组合列表

供油箱数	供油组合
单箱供油（4 种）	【2】、【3】、【4】、【5】
双箱供油（14 种）	【1，2】、【1，3】、【1，4】、【1，5】、 【2，3】、【2，4】、【2，5】、【2，6】、 【3，4】、【3，5】、【3，6】、 【4，5】、【4，6】、 【5，6】
三箱供油（16 种）	【1，2，3】、【1，2，4】、【1，2，5】、【1，2，6】、 【1，3，4】、【1，3，5】、【1，3，6】、 【1，4，5】、【1，4，6】、 【1，5，6】、 【6，2，3】、【6，2，4】、【6，2，5】、 【6，3，4】、【6，3，5】、 【6，4，5】

如果不考虑上述“供油时间”约束和“供油速度”约束，即公式（36）~(40)，则每一秒有

34 种供油组合，7200 秒内则有34^{7200}种情况，这是一个指数级别的计算规模，难以准确求得模型最优解。因此本文设计了近似解法，可以在尽可能短的时间内得到近似最优解。

6.3.1 模型假设

（1）“油箱供油时长”假设：每个油箱每次供油的持续时长为T（T为飞行时长 7200 的因数），即每个油箱每次供油的时长不大于T也不小于T；

（2）“油箱供油速度”假设：为了避免 2 号油箱和 5 号油箱无油，本文假设，如果 1 号油箱和 2 号油箱同时供油，且 2 号油箱供油速度不超过 1 号油箱的最大供油速度，则 1 号油箱会以 2 号油箱的供油速度供油，否则 1 号油箱以最大供油速度供油。5 号油箱和 6 号油箱同理；

（3）“油箱供油比例”假设：当有多个油箱为发动机供油时，本文假设这两个油箱在这段时间内会按照固定的某个比例同时供油；

（4）“油箱供油总量”假设：油箱不会多供油，即向发动机供油的量与发动机消耗的油量相等。

6.3.2 算法思想--贪婪策略

本文的算法核心为：贪心[4]。贪心算法可以获得局部最优解，但不一定能得到模型最优解，关键是贪心策略的选择。

根据上述约束条件和目标函数，本文制定的**贪心策略**为：在每一个T时间段内，选择一个最优的供油组合和供油速度，使得该时间段内质心欧式距离的最大值最小，直到飞行结束。

算法实现的关键步骤如下：

表 5：基于贪心思想的飞行器供油策略的主要思路

基于贪心思想的飞行器供油策略主要思路
（1）选择T，满足：不小于 60 且为 7200 的因数列表$List$；
（2）选择 34 种供油组合$Groups$中的一种；
（3）将选中的油箱设置为 1，未选中的油箱设置为 0；
（4）判断该组合是否为单箱供油，如果是，则设定该油箱的供油速度$S_k(t)$等于发动机耗油速度$S_{engine}(t)$；如果否，则按照固定比例对组合油箱的供油速度进行分配；
（5）检查该速度是否违背油箱“供油速度约束”的 2 个子项（四个主油箱的供油速度之和大于或等于飞行器所需耗油的速度；每个油箱的供油速度不能超过该油箱参数所规定的输送油量上限）；
（6）检查供油期间所剩油量是否不小于 0；
（7）如果上述约束条件都满足，则计算该种供油组合情况下的每个时刻的欧式距离的最大值；

（8）输出欧式距离最大值的最小值。

伪代码如下所示：

表 6：基于贪心思想的飞行器供油策略伪代码

基于贪心思想的飞行器供油策略伪代码

for T in $List$:
    for item in $Groups$：
        选中的油箱: O_k=1　or　未选中: $O_k = 0$；
        if 单箱供油：
            $S_k(t) = S_{engine}(t)$
        else:
            $S_k(t) = ratio * S_{engine}(t)$
        if sum ($S_k(t)$)< $S_{engine}(t)$ or $S_k(t) > S_{max}(k)$ or $M_k(t) < 0$:
            break
        else:
            求解得$D(t)$
    计算 min($D(t)$)
计算 max(min($D(t)$))

6.4 模型结果

6.4.1 结果分析

本文使用 Python 语言实现了上述模型约束，并用贪心思想进行了模型求解。本文将满足假设条件“不小于 60 且为 7200 的因数”的所有T情况进行代入求解，用算法计算出质心欧式距离的最大值，得到每个T时间段的局部最优解。

实验发现：过长的持续供油时间会导致质心失衡，T的合适范围为 60～90。在 60～90 范围内，本文使用上述算法计算出质心欧式距离的最大值。结果如图 14 所示，**当$T = 85$时，质心欧式距离的最大值达到最小。**

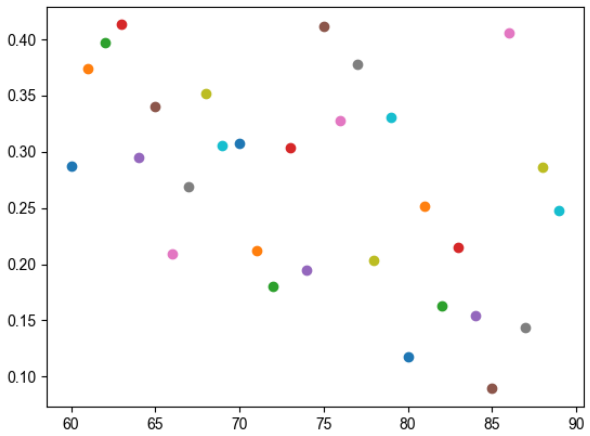
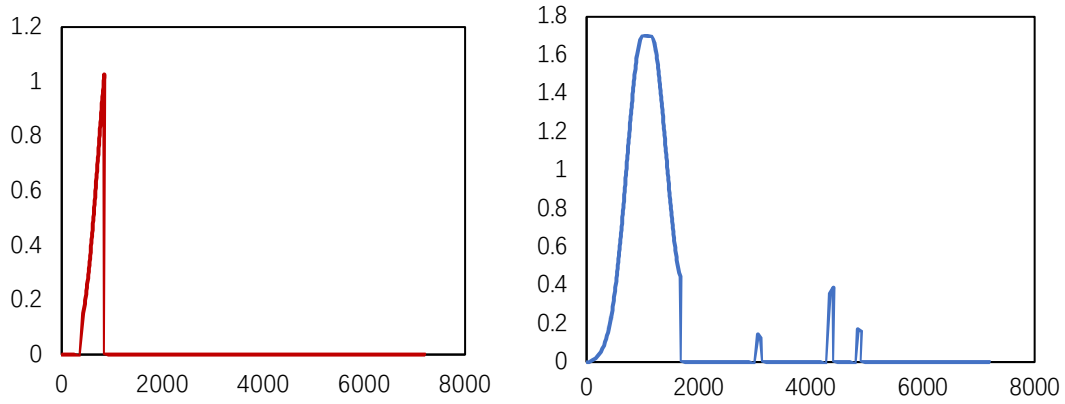


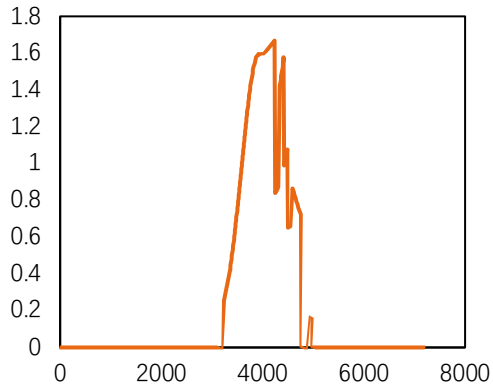
图 14：不同T取值下的质心欧式距离的最大值散点图

$T = 85$时，6 个油箱各自的供油曲线如图 15（a）~（f）所示，汇总曲线如图 16 所示。

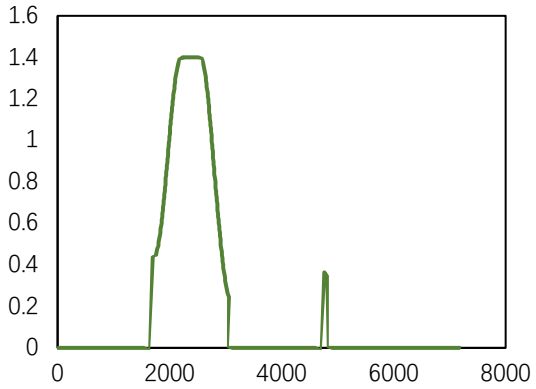


（a）

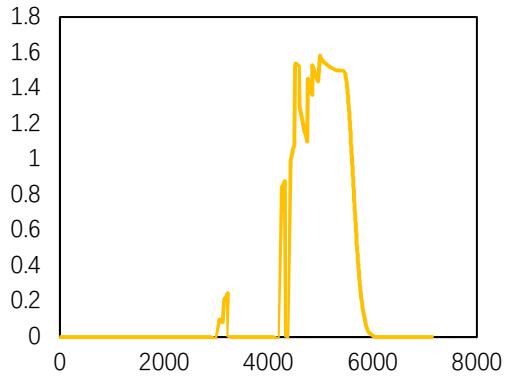
（b）



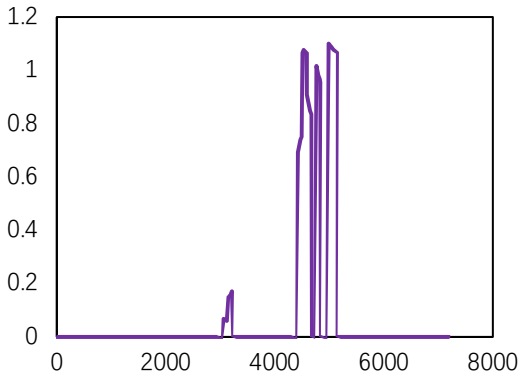
（c）



（d）



（e）



（f）

图 15：（a）~（f）分别对应 1～6 号的 6 个油箱各自的供油速度曲线

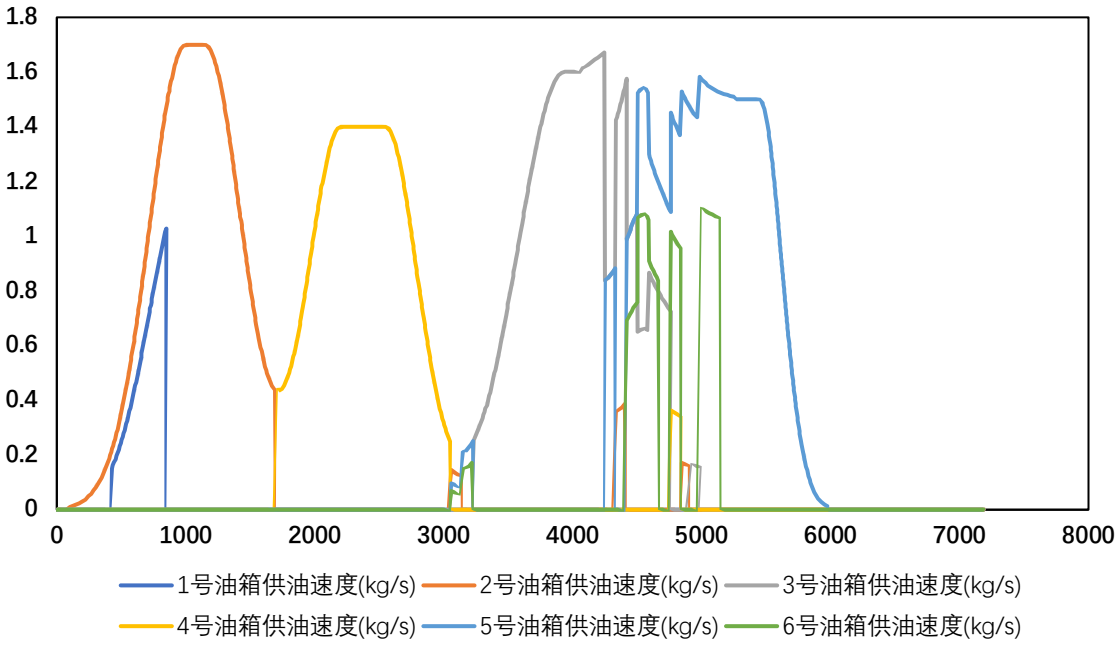


图 16：6 个油箱的供油曲线汇总图

4 个主油箱的总供油曲线如图 17 所示：

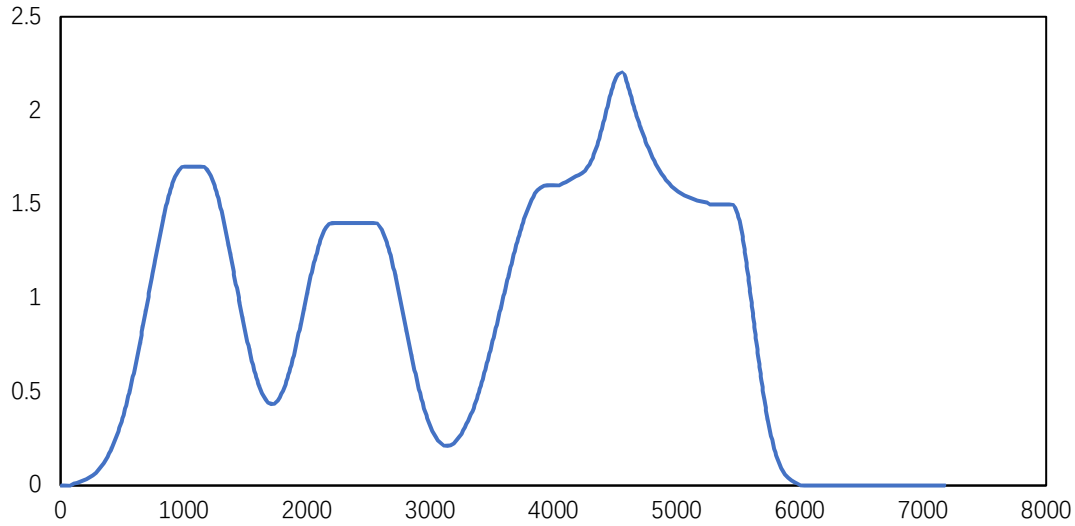


图 17：四个主油箱的供油速度曲线图

飞行器瞬时质心与理想质心距离的最大值为 0.089183 米，此时飞行器飞行 4982 秒，质心点的坐标为(-1.8139, -0.0678, 0.0316)。

4 个油箱的总供油量为 6441.52421 千克，此供油量正好等于飞行器所需的耗油量。

6.4.2 结果总结

综上所述：针对问题二中的各个问题，我们采用了贪婪算法求解满足约束条件下，质心位置距理想质心欧式距离最小的 6 个油箱供油策略，据此求得的结果如表 7。

表 7：问题二结果汇总

要求	答案
给出飞行器飞行过程中 6 个油箱各自的供油速度曲线；	**图 15**
给出飞行器飞行过程中 4 个主油箱的总供油速度曲线；	**图 17**
飞行器瞬时质心与理想质心距离的最大值；	**0.0891m**
给出 4 个主油箱的总供油量；	**6441.52421kg**
将 6 个油箱的供油速度数据按时间顺序存入表中。	**已完成**

6.5 模型有效性验证

模型验证主要体现在两个方面：

（1）各个油箱的供油速度是否超过该油箱参数中所规定的输送油量上限；

（2）任意时刻下 4 个主油箱的总供油速度是否大于等于飞机的耗油速度；

为了验证（1），从图 15、表 8 中可得 1~6 号油箱的实际供油速度分别为 1.1、1.7、1.69945、1.4、1.59643、1.1,均不大于附件一中给出的油箱的最大供油速度限制 1.1、1.8、1.7、1.5、1.6、1.1。

表 8：实际供油速度与油箱供油速度上限的对比情况

对比主题	飞行中油箱最大供油速度（单位为：千克/秒）					
实际供油	1.1	1.7	1.69945	1.4	1.59643	1.1
供油上限	1.1	1.8	1.7	1.5	1.6	1.1

为了验证（2），从图 18 中可以看出，飞行器的耗油速度曲线与 4 个主油箱的供油速度曲线完全重合，即飞行器能一直获得充足的油，我们的模型可靠性非常好。需要说明的是，算法中也兼顾了供油速度的平滑性。

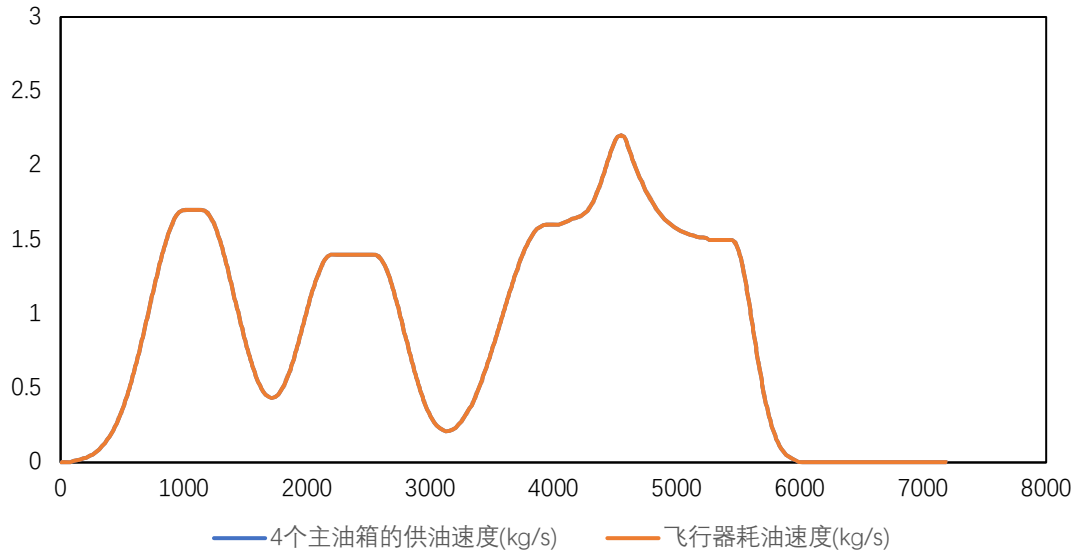


图 18：四个主油箱供油与飞行器耗油的速度对比曲线图

七、问题三的求解

7.1 问题分析

已知某次任务的飞行器计划耗油速度数据和飞行器在飞行器坐标系下的理想质心位置数据。在飞行器始终保持平飞的任务规划中，问题三要求制定满足基本假设条件的 6 个油箱初始载油量及供油策略，目标是使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。初步分析，该问题仍然是一个目标优化问题。

7.2 模型建立

7.2.1 约束条件

（1）“剩余油量”约束：问题三在问题二的约束条件上新增了一个条件：初始油量未知，在任务结束时，6 个油箱中的剩余燃油总量至少有 $1m^3$。已知油的密度为 $850kg/m^3$，所以该约束等价于在 7200 秒时，6 个油箱中的剩余燃油总量的质量至少有 $850kg$。第k个油箱分配得到的油量为M_k，因此该目标可在数学上表达为：

$$\sum_{k=1}^{6} M_k(7200) \geq 850 \tag{49}$$

本文基于这样的考虑：6 个油箱中油的总重量至少为飞行器在这段时间内的总耗油量与最低剩余油量的质量只和。假设油箱中的初始总油量为M_0，需要设计一个油量分配方案，将油分配到各个油箱中。在分配时，需要考虑以下约束：

（2）“油箱分配上限”约束：各个油箱分配到的油量不能超过油箱的最大总油量。已知油的密度为 $850kg/m^3$，第k个油箱的长宽高分别为a_k、b_k、c_k，因此该约束可表达为：

$$M_k \leq 850 * a_k * b_k * c_k \tag{50}$$

（3）“油量分配总和”约束：初始分配时，分配到各个油箱中的油量总和应该等于初始总油量。表达为：

$$\sum_{k=1}^{6} M_k = M_0 \tag{51}$$

（4）“初始质心”约束：分配后，飞行器的质心要尽可能地接近 0 时刻的理想飞行器质心。在 0 时刻飞行器的理想质心为(x_0, y_0, z_0)，飞行器质量为m_0，第k个油箱在x、y、z轴方向的中心位置分别表示为x_k、y_k、z_k。根据质心公式，可得到以下 3 个方程来满足该约束条件：

$$\frac{\sum_{k=1}^{6} M_k x_k}{m_0 + M_0} = x_0 \tag{52}$$

$$\frac{\sum_{k=1}^{6} M_k y_k}{m_0 + M_0} = y_0 \tag{53}$$

$$\frac{\sum_{k=1}^{6} M_k\left(z_k+\frac{\frac{M_k}{850*a_k*b_k}-c_k}{2}\right)}{m_0+M_0}=z_0 \tag{54}$$

通过以上约束条件和方程，本文可得油量分配的近似最优解，使得初始质心尽量接近0时刻的理想质心。

7.2.2 目标函数

（1）目标一：

为了让按初始油量分配策略分配得到的飞行器质心，与0时刻飞行器理想质心的欧式距离尽可能的接近，该目标可表达为：

$$D(0)=\min_{M_1,\ldots,M_6}\|\overrightarrow{c_1}(0)-\overrightarrow{c_2}(0)\|_2$$

$$s.t.\begin{cases}\sum_{k=1}^{6} M_k(7200)\geq 850\\ M_k\leq 850*a_k*b_k*c_k\\ \sum_{k=1}^{6} M_k=M_0\\ \overrightarrow{c_1}(0)=\left\{\frac{\sum_{k=1}^{6} M_k x_k}{m_0+M_0},\ \frac{\sum_{k=1}^{6} M_k y_k}{m_0+M_0},\ \frac{\sum_{k=1}^{6} M_k\left(z_k+\frac{\frac{M_k}{850*a_k*b_k}-c_k}{2}\right)}{m_0+M_0}\right\}\\ \overrightarrow{c_2}(0)=\{x_0,\ y_0,\ z_0\}\end{cases} \tag{55}$$

（2）目标二：

在飞行器保持平飞的任务规划中，目标是使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。可数学表达为：

$$D(t)=\min\max_{t}\|\overrightarrow{c_1}(t)-\overrightarrow{c_2}(t)\|_2 \tag{56}$$

$$s.t.\begin{cases} \sum_{k=1}^{6} M_k(7200) \geq 850 \\ M_k \leq 850 * a_k * b_k * c_k \\ \sum_{k=1}^{6} M_k = M_0 \\ \frac{\sum_{k=1}^{6} M_k x_k}{m_0 + M_0} = x_0 \\ \frac{\sum_{k=1}^{6} M_k y_k}{m_0 + M_0} = y_0 \\ \frac{\sum_{k=1}^{6} M_k \left(z_k + \frac{M_k}{850 * a_k * b_k} - c_k \right)}{m_0 + M_0} = z_0 \end{cases} \tag{57}$$

7.3.3 单目标优化模型

（1）针对目标一：差分进化算法

差分进化算法（简称 DE 算法)[1] ，它是一种基于群体间个体差异的随机启发式搜索算法。本文使用差分进化算法，可制定好的初始油量分配策略。

差分优化模型的步骤如下：

表 9：差分优化模型核心步骤

差分优化模型核心步骤
（1） 生成初始种群；
（2） 变异操作。从当前代群体中随机选择若干个染色体形成新的变异矢量；
（3） 交叉操作。种群交叉增加群体的多样性；
（4） 选择操作。向量和目标向量对评价函数进行比较，循环产生的最终子代个体。
（5） 重复执行变异、交叉和选择操作，直到达到最大进化代数。

（2）针对目标二：同问题二贪心算法

分析约束条件和目标函数后，可发现制定好初始油量的分配策略后，通过问题二中的供油策略选择算法，可以求得一套近似最优的供油策略，并得到质心欧式距离最大值的最小值。

与问题二不同的是，问题三可通过上述差分进化算法得到初始油量分配策略。

7.3 模型求解

7.3.1 算法实现的关键步骤

表 10：问题三求解算法步骤

问题三求解算法步骤
（1）通过差分优化算法得到单目标优化的结果，作为初始油量分配方案；

（2）将初始油量分配方案带入问题一的算法中，计算每个 T 时间内的最优供油组合和供油速度，使得该段时间内的质心欧式距离的最大值最小；

（3）得到每个 T 时间段内的局部最优解，作为近似解；

（4）用多个不同 M_0 取值进行实验，求得每个M_0对应的质心欧式距离最大值的最小值，选择较小的质心距离对应的初始油量分配方案作为最佳分配方案。

7.3.2 拟合不同M_0取值下的欧式距离最大值

然后我们再用多个不同 M_0 取值进行实验，求得每个M_0对应的质心欧式距离最大值的最小值，实验结果如图 19 所示：

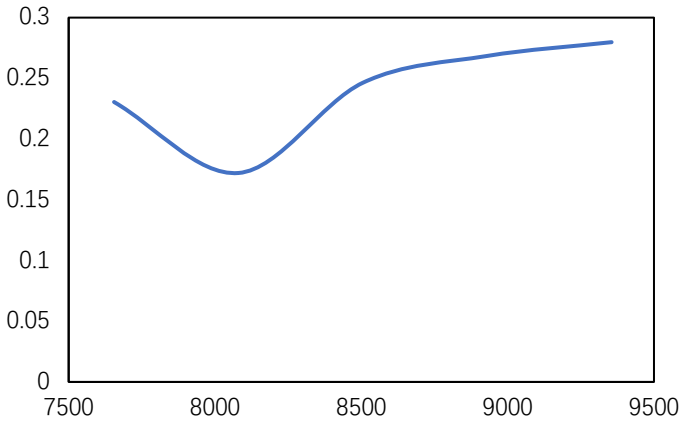


图 19：不同M_0对应的质心欧式距离最大值的最小值曲线图

表 11：不同M_0取值下的欧式距离最大值

初始油量（kg）	质心欧式距离的最大值
7655.17467	0.230492
7655.17467+0.5*850	0.172049
7655.17467+1*850	0.246351
7655.17467+1.5*850	0.268438
7655.17467+2*850	0.279779

表 11 展示了不同M_0取值下的欧式距离最大值，可以看出**初始油总量为7655.17467+0.5*850=8080.1467 时产生了最佳的分配方案，**此时飞行器质心与理想质心距离的最大值为 0.172049 米。

7.4 模型结果

7.4.1 结果分析

通过对上述模型的求解，可得到 6 个油箱的初始载油量（单位为千克），如表 12 所示：

表 12：6 个油箱的初始载油量表

油箱编号	1	2	3	4	5	6
初始载有量	300	1645	1613.0	1684.1	2434.7	403.4

6 个油箱供油速度曲线如图 20（a）~（f）所示：

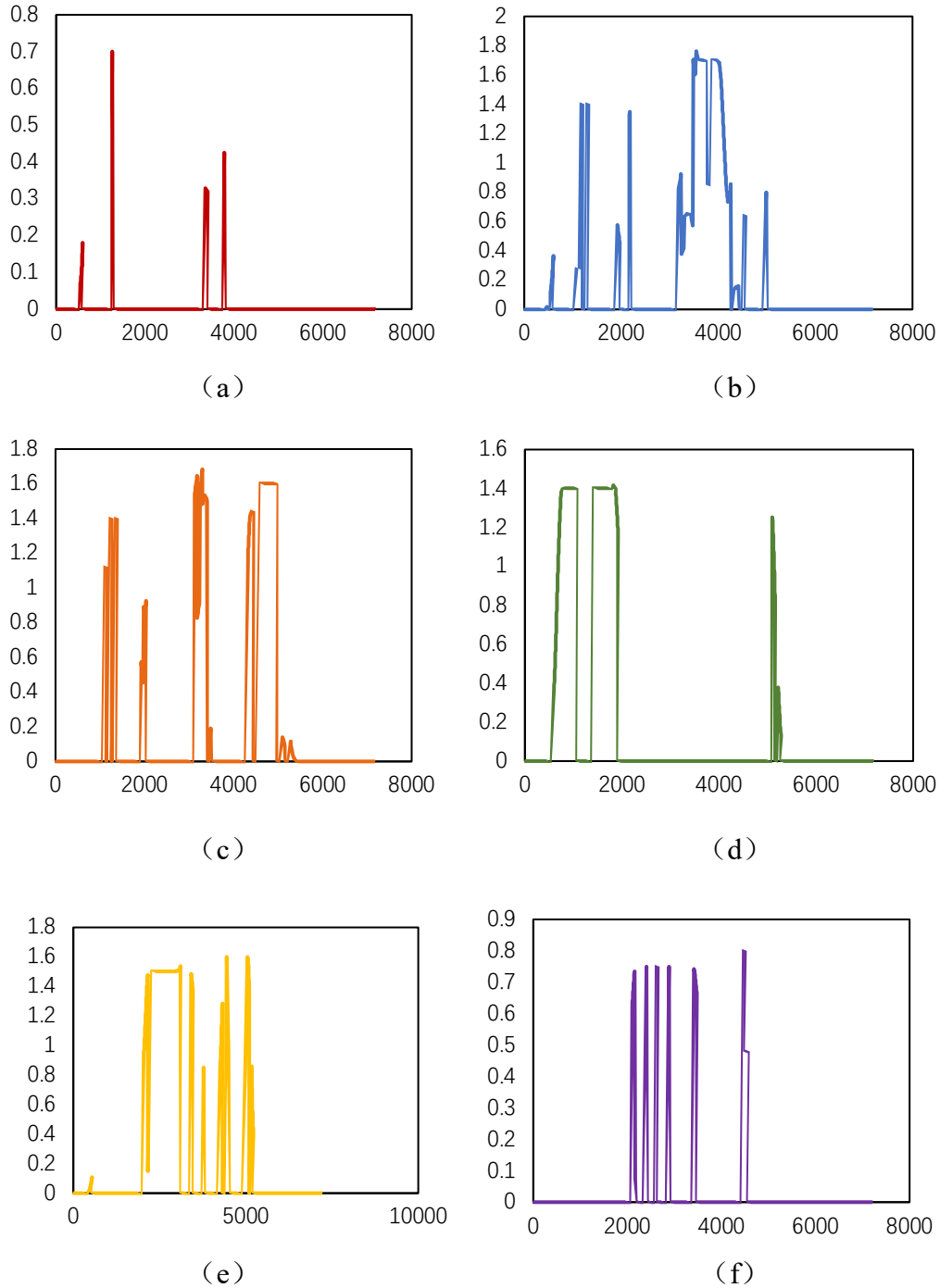


图 20：（a）~（f）分别对应 1～6 号的 6 个油箱各自的供油速度曲线

4 个主油箱的总供油曲线如图 21 所示：

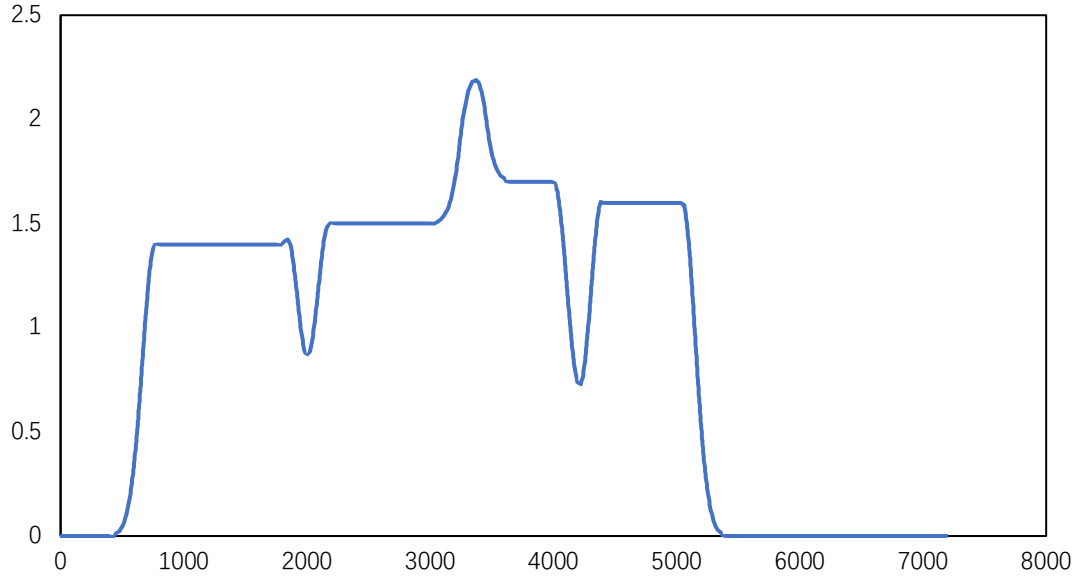


图 21：四个主油箱的总供油曲线图

飞行器瞬时质心与飞行器（不载油）质心$\vec{c}_0$的最大距离偏差 0.172049 米，此时飞行器飞行 3349 秒，质心点的坐标为(0.5786, -0.1050, -0.0433)。

4 个主油箱的总供油量 6805.17467 千克，此供油量正好等于飞行器所需的耗油量。

7.4.2 结果总结

综上所述：针对问题三中的各个问题，我们制定了满足约束条件的 6 个油箱初始载油量及供油策略，使得结束时 6 个油箱剩余燃油总量至少 $1m^3$，并且飞行器每一时刻的质心位置与理想质心位置的欧氏距离的最大值达到最小，据此求得的结果如表 13 所示：

表 13：问题三结果表

要求	答案
给出 6 个油箱的初始载油量；	[300，1645，1613.0，1684.1，2434.7，403.4]
给出 6 个油箱的供油速度曲线；	图 20
给出 4 个主油箱的总供油速度曲线；	图 21
飞行器质心与理想质心距离的最大值；	0.172049m
给出 4 个主油箱的总供油量；	6805.17467kg
将 6 个油箱的初始油量存入结果表中；	已完成
将 6 个油箱的供油速度数据按时间存入结果表中；	已完成

7.5 模型有效性验证

模型验证主要体现在以下三个方面：

（1）各个油箱被分配到的初始油量是否超过该油量的最大存储量。

（2）各个油箱的供油速度是否超过该油箱参数中所规定的输送油量上限；

（3）任意时刻下 4 个主油箱的总供油速度是否大于等于飞机的耗油速度；

为了验证（1）

为了验证（2），从图 20、表 14 中可得 1~6 号油箱的实际供油速度分别为 0.7、1.763、1.691、1.425、1.6、0.8（单位为 kg/s），均不大于附件一中给出的油箱的最大供油速度限制 1.1、1.8、1.7、1.5、1.6、1.1。

表 14：实际供油速度与油箱供油速度上限的对比情况

对比主题	**飞行中油箱最大供油速度（单位为：千克/秒）**					
实际供油	0.7	1.763	1.691	1.425	1.6	0.8
供油上限	1.1	1.8	1.7	1.5	1.6	1.1

为了验证（3），从图 22 中可以看出，飞行器的耗油速度曲线与 4 个主油箱的供油速度曲线完全重合，即飞行器能一直获得充足的油，我们的模型可靠性非常好。需要说明的是，算法中也兼顾了供油速度的平滑性。

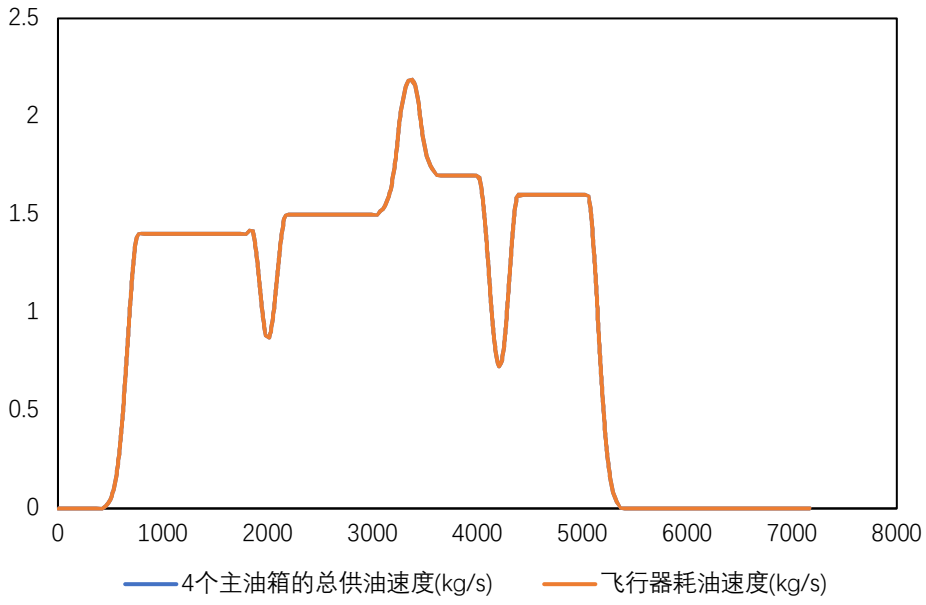


图 22：4 个主油箱的总供油曲线与计划耗油速度曲线对比图

八、问题四的求解

8.1 问题分析

在实际任务规划过程中，飞行器俯仰角随时间变化而变化。已知飞行器俯仰角的变化数据与耗油速度数据，问题四要求制定油箱供油策略，目标仍是使得飞行器每一时刻质心位置$\overrightarrow{c_1}(t)$与理想质心位置$\overrightarrow{c_2}(t)$的欧氏距离的最大值达到最小。

8.2 模型建立

本问题在问题二的基础上增加了一个约束条件：飞行器的俯仰角θ会随时间变化，另外本问题将飞行器的质心固定为飞行器不载油时的质心 $\vec{c}_0 = (0,0,0)$。

本文的算法核心仍然为：贪心。贪心算法可以获得局部最优解，但不一定能得到模型最优解，关键是贪心策略的选择。

8.3 模型求解

模型求解时仍使用问题二提出的供油策略求解模型和算法，但有以下不同：

（1）在计算欧式距离时，飞行器的理想质心坐标不再由附件读入，而是替换为$\vec{c}_2 = (0,0,0)$；

（2）引入了俯仰角，俯仰角是从附件中读入的随时间变化的函数θ_t，因此计算飞行器质心坐标需要使用问题一中求得的包含俯仰角度的质心计算公式。

目标函数表示如下：

$$\| \overrightarrow{c_1}(t) - \overrightarrow{c_2}(t) \|_2 = \sqrt{(x_1(t)-0)^2 + (y_1(t)-0)^2 + (z_1(t)-0)^2}$$

其中：

$$\overrightarrow{c_1}(t) = \frac{m_0 * \vec{0} + \sum_{i=1}^{6} M_i(t)\,\overrightarrow{Q_i}(t)}{m_0 + \sum_{i=1}^{6} M_i(t)}$$

算法流程如下：

表 15：基于贪心思想的飞行器供油策略主要思路

基于贪心思想的飞行器供油策略主要思路
（1）输入当前时刻 6 个油箱的质量和俯仰角；
（1）选择T，满足：不小于 60 且为 7200 的因数列表$List$；
（2）选择 34 种供油组合$Groups$中的一种；
（3）将选中的油箱设置为 1，未选中的油箱设置为 0；
（4）判断该组合是否为单箱供油，如果是，则设定该油箱的供油速度$S_k(t)$等于发动机耗油速度$S_{engine}(t)$；如果否，则按照固定比例对组合油箱的供油速度进行分配；
（5）检查该速度是否违背油箱“供油速度约束”的 2 个子项（四个主油箱的

供油速度之和大于或等于飞行器所需耗油的速度；每个油箱的供油速度不能超过该油箱参数所规定的输送油量上限）；

（6）检查供油期间所剩油量是否不小于 0；

（7）如果上述约束条件都满足，则计算该种供油组合情况下的每个时刻的欧式距离的最大值，在计算时，使用（0，0，0）作为理想质心，使用问题一中解得的与θ和 6 个油箱质量有关的质心坐标计算公式来计算欧式距离；

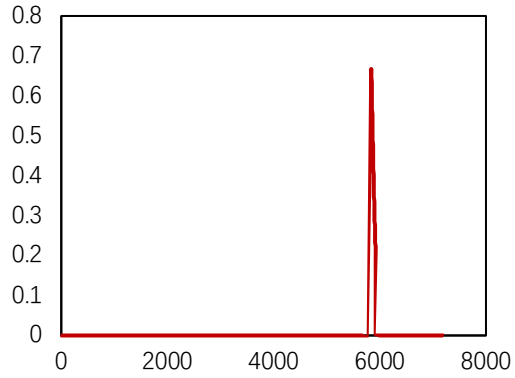
（8）输出欧式距离最大值的最小值；

8.4 模型结果

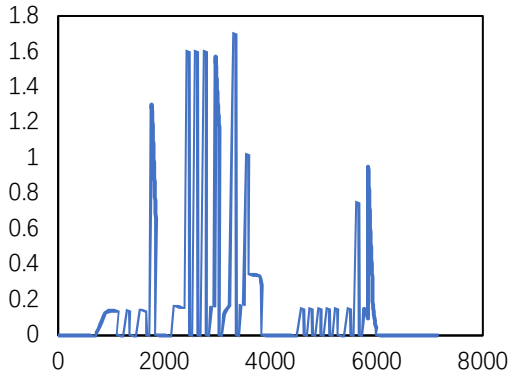
8.4.1 结果分析

本文使用 Python 语言实现了上述模型，结合使用了问题二和问题三的主要核心算法。实验发现：**当$T = 67$时，质心欧式距离的最大值达到最小。**

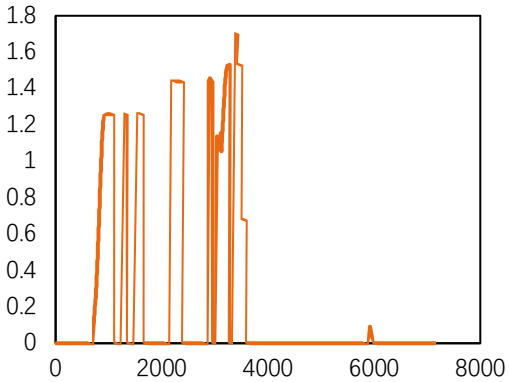
6 个油箱供油速度曲线如图 23（a）~（f）所示：



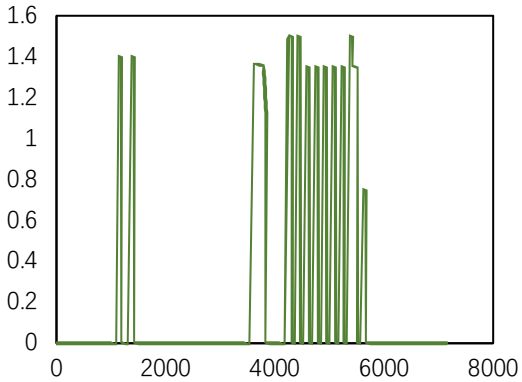
（a）



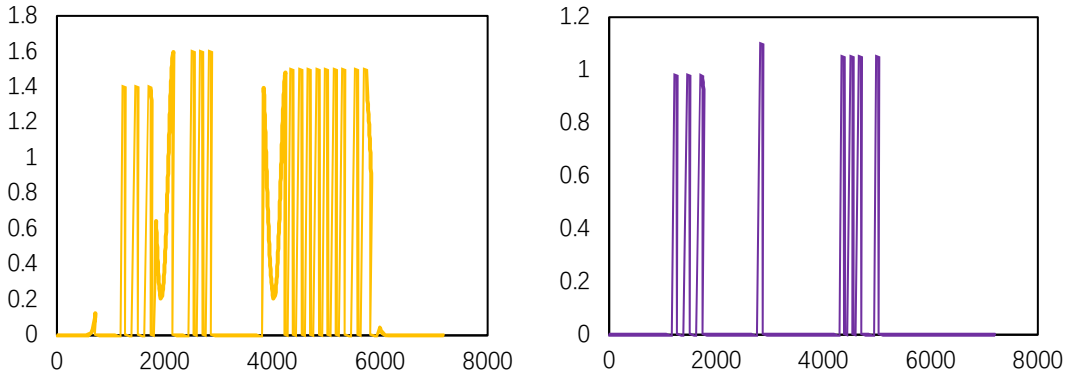
（b）



（c）



（d）



（e）　　（f）

图 23：（a）~（f）分别对应 1～6 号的 6 个油箱各自的供油速度曲线

4 个主油箱的总供油曲线与计划耗油速度曲线对比如图 24 所示：

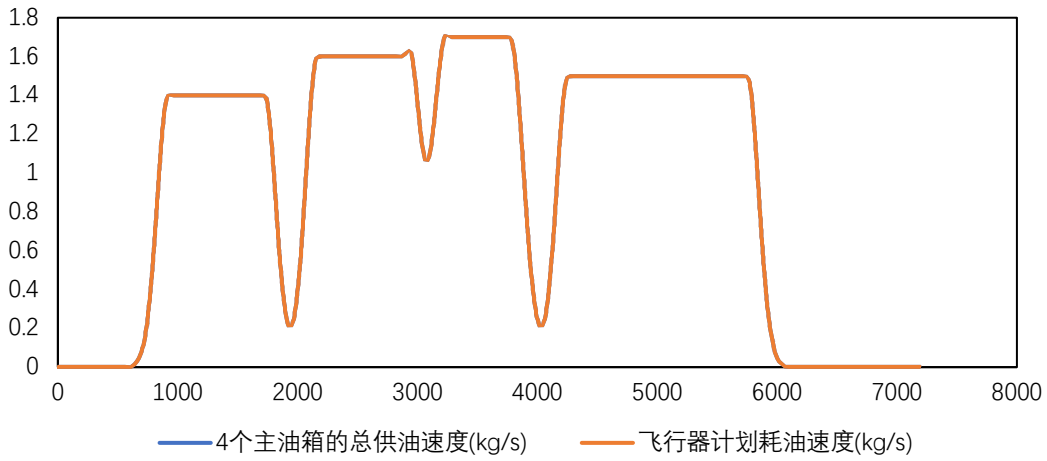


图 24：4 个主油箱的总供油曲线与计划耗油速度曲线对比图

飞行器瞬时质心与飞行器（不载油）质心$\vec{c}_0$的最大距离偏差 0.241299 米，此时飞行器飞行 3840 秒，质心点的坐标为(-0.2345, -0.0566, -0.0046)。

4 个主油箱的总供油量 7035.54516 千克，此供油量正好等于飞行器所需的耗油量。

8.4.2 结果总结

综上所述：针对问题四中的各个问题，求得的结果如表 16。

表 16：问题四结果汇总

要求	答案
飞行器飞行过程中 6 个油箱各自的供油速度曲线；	**图 23**
4 个主油箱的总供油曲线与计划耗油速度曲线；	**图 24**
飞行器瞬时质心与飞行器不载油质心$\vec{c}_0$的最大距离偏差	**0.241299m**
给出 4 个主油箱的总供油量；	**7035.54516kg**
将 6 个油箱的供油速度数据按时间顺序存入表中。	**已完成**

8.5 模型有效性验证

模型验证主要体现在以下两个个方面：

（1）各个油箱的供油速度是否超过该油箱参数中所规定的输送油量上限；

（2）任意时刻下 4 个主油箱的总供油速度是否大于等于飞机的耗油速度；

为了验证（1），从图 23、表 17 中可得 1~6 号油箱的实际供油速度分别为 0.66715678、1.7、1.7、1.5、1.6、1.1（单位为 kg/s），均不大于附件一中给出的油箱的最大供油速度限制 1.1、1.8、1.7、1.5、1.6、1.1。

表 17：实际供油速度与油箱供油速度上限的对比情况

对比主题	飞行中油箱最大供油速度（单位为：千克/秒）					
实际供油	0.667	1.7	1.7	1.5	1.6	1.1
供油上限	1.1	1.8	1.7	1.5	1.6	1.1

为了验证（2），从图 24 中可以看出，飞行器的耗油速度曲线与 4 个主油箱的供油速度曲线完全重合，即飞行器能一直获得充足的油，我们的模型有效性非常好。需要说明的是，算法中也兼顾了供油速度的平滑性。

九、模型评价

本文建立的模型针对质心计算问题采用了分类讨论策略，对各种情况进行了详细的分析与推导，拥有坚实的物理与数学基础。相对于计算积分的方法，本模型的算法的复杂度较低，时间复杂度为 O(n)，空间复杂度为 O(1)，并且更加可靠与快速，为类似的问题提供了新的思路。

针对策略优化等问题，本文采用了单目标优化模型[2]，利用了差分优化和贪心算法等策略，将指数级规模的问题进行简化从而得到局部最优解，最后组成全局近似最优解并进行问题解答，算法的时间复杂度为 O(n)，空间复杂度为 O(1)，能够更快更好地求得近似最优解。

最终的结果也表明本模型效果优秀，效率极高。

十、参考文献

[1] 张晖, 王水清. 差分优化算法及其应用[J]. 科技视界, 2017, 000(008):107-107.

[2] 苏勇彦. 单目标、多目标优化进化算法及其应用[D]. 武汉理工大学, 2007.

[3] 王明美. 平面组合图形质心的计算[J]. 合肥师范学院学报, 2011.

[4] 陆焱. 浅谈算法设计技术——贪心策略[J]. 电脑知识与技术, 2009, 005(020):5485-5486,5489.

十一、附件

10.1 问题一算法求解代码

```
import xlrd
import xlwt
import math
#注：油箱 1 输送给油箱 2，油箱 6 输送给油箱 5，油箱 2，3，4，5 燃烧燃料



record_cal = [0,0,0,0,0,0,0,0,0,0]
#xyz 和 zbc 都以 m 为单位
box1_xyz = [8.91304348, 1.20652174, 0.61669004]
box2_xyz = [6.91304348, -1.39347826, 0.21669004]
box3_xyz = [-1.68695652, 1.20652174, -0.28330996]
box4_xyz = [3.11304348, 0.60652174, -0.18330996]
box5_xyz = [-5.28695652, -0.29347826, 0.41669004]
box6_xyz = [-2.08695652, -1.49347826, 0.21669004]

box_xyzs = [box1_xyz, box2_xyz, box3_xyz, box4_xyz, box5_xyz, box6_xyz]

box1_abc = [1.5, 0.9, 0.3]
box2_abc = [2.2, 0.8, 1.1]
box3_abc = [2.4, 1.1, 0.9]
box4_abc = [1.7, 1.3, 1.2]
box5_abc = [2.4, 1.2, 1]
box6_abc = [2.4, 1, 0.5]

box_abcs = [box1_abc, box2_abc, box3_abc, box4_abc, box5_abc, box6_abc]

box_xyz0s = []
for j in range(6):
    box_abc = box_abcs[j]
    box_xyz = box_xyzs[j]
    box_x0 = box_xyz[0]- 0.5 * box_abc[0]
```

```
            box_y0 = box_xyz[1] - 0.5 * box_abc[1]
            box_z0 = box_xyz[2] - 0.5 * box_abc[2]
            box_xyz0s.append([box_x0,box_y0,box_z0])



    planet_m0 = 3000 #kg
    ru = 850 #kg/m^3
    box_vs = [0.3, 1.5, 2.1, 1.9, 2.6, 0.8] #m^3
    box_ms = []
    for i in range(6):
        box_ms.append(box_vs[i]*ru)

    box_mass_centers = [[],[],[],[],[],[]]

    def calMassCenterOfSingleBox(h, theta, a, b ,c):
        print('tan_theta:'+str(math.tan(abs(((theta/180)*math.pi)))))
        if theta == 0:#矩形
            print('矩形')
            record_cal[0] = record_cal[0] + 1
            return [a/2,b/2,h/2]
        if theta >0 :
            tan_theta =    math.tan((theta/180)*math.pi)
            # if a*tan_theta/2 < h and a*tan_theta/2 <= (c-h): #梯形 1-正
            if (h>c/2 and tan_theta <= (c-h)*2/a) or (h<=c/2 and
tan_theta<(h*2)/a):
                print('梯形 1-正')
                record_cal[1] = record_cal[1] + 1
                mass_center_rect = [a/2, b/2 ,h/2-a*tan_theta/4]
                mass_center_tria = [a/3, b/2, h-a*tan_theta/6]
                m_rect = (h-a*tan_theta/2)*a
                m_tria = a*a*tan_theta/2
                # print(str(m_rect + m_tria)+' '+str(a*h))
                mass_center_sum = [0,0,0]
                for i in range(3):
```

```
                    mass_center_sum[i] = mass_center_rect[i]*m_rect + mass_center_tria[i]*m_tria
                    return [k/(m_rect+m_tria) for k in mass_center_sum]


            if h<=c/2 and tan_theta>=(h*2)/a and tan_theta<=pow(c,2)/(2*a*h): #三角形 1-正
                    print('三角形 1-正')
                    record_cal[2] = record_cal[2] + 1
                    mass_center_tri = [pow(2*a*h/tan_theta,0.5)/3, b/2, pow(2*a*h*tan_theta,0.5)/3]
                    return mass_center_tri
            if h>c/2 and tan_theta>(c-h)*2/a and tan_theta<pow(c,2)/(2*a*(c-h)): #五边形 1-正
                    print('五边形 1-正')
                    record_cal[3] = record_cal[3] + 1
                    x = a - pow(2*a*(c-h)/tan_theta,0.5)
                    mass_center_rect1 = [x/2, b/2, c/2]
                    m_rect1 = x*c
                    mass_center_rect2 = [x/2+a/2, b/2, (c-(a-x)*tan_theta)/2]
                    m_rect2 = (a-x)*(c-(a-x)*tan_theta)
                    mass_center_tria = [(a+2*x)/3, b/2, c-2*(a-x)*tan_theta/3]
                    m_tria = pow(a-x,2)*tan_theta/2
                    # print(str(m_rect1 + m_rect2 + m_tria) + ' ' + str(a * h))
                    mass_center_sum = [0, 0, 0]
                    for i in range(3):
                        mass_center_sum[i] = mass_center_rect1[i] * m_rect1 + mass_center_rect2[i] * m_rect2 +mass_center_tria[i] * m_tria
                    return [k/(m_rect1+m_rect2+m_tria) for k in mass_center_sum]



            # if (h<=c/2 and a*tan_theta/2>=h and 2*tan_theta*a*h>pow(c,2)) or (h>c/2 and (h+a*tan_theta/2) >= c and tan_theta*2*a*(c-h)>=pow(c,2)): #梯形 2-正
            if (h <= c / 2 and tan_theta > pow(c, 2)/(2*a*h)) or (h > c / 2 and tan_theta >= pow(c, 2)/(2 * a * (c - h))):
```

```
                    print('梯形 2-正')
                    record_cal[4] = record_cal[4] + 1
                    # mass_center_rect = [a*h/2*c-c/4*tan_theta, b/2, c/2]
                    x = a*h/c-c/(2*tan_theta)
                    mass_center_rect = [x / 2 , b / 2, c / 2]
                    m_rect = a*h - pow(c,2)/(2*tan_theta)
                    mass_center_tria = [x + c/(3*tan_theta), b/2, c/3]
                    m_tria = pow(c,2)/(2*tan_theta)
                    # print(str(m_rect + m_tria) + ' ' + str(a * h))
                    mass_center_sum = [0, 0, 0]
                    for i in range(3):
                        mass_center_sum[i] = mass_center_rect[i] * m_rect + mass_center_tria[i] * m_tria
                    return [k/(m_rect+m_tria) for k in mass_center_sum]

                # return [a / 2, b / 2, h / 2]



            if theta <0 :
                theta = abs(theta)
                tan_theta = math.tan(((theta/180)*math.pi))
                if (h>c/2 and tan_theta <= (c-h)*2/a) or (h<=c/2 and tan_theta<(h*2)/a):
                # if a * tan_theta / 2 < h and a * tan_theta / 2 <= (c - h):
                    print('梯形 1-负')
                    record_cal[5] = record_cal[5] + 1
                    mass_center_rect = [a / 2, b / 2, h / 2 - a * tan_theta / 4]
                    mass_center_tria = [2*a / 3, b / 2, h - a * tan_theta / 6]
                    m_rect = (h - a * tan_theta / 2) * a
                    m_tria = a * a * tan_theta / 2
                    mass_center_sum = [0, 0, 0]
                    for i in range(3):
                        mass_center_sum[i] = mass_center_rect[i] * m_rect + mass_center_tria[i] * m_tria
```

```
                return [k/(m_rect+m_tria) for k in mass_center_sum]

            if h<=c/2 and tan_theta>=(h*2)/a and tan_theta<=pow(c,2)/(2*a*h):
                print('三角形 1-负')
                record_cal[6] = record_cal[6] + 1
                mass_center_tri = [a-(pow(2*a*h/tan_theta,0.5)/3), b/2,
pow(2*a*h*tan_theta,0.5)/3]
                return mass_center_tri

            if h>c/2 and tan_theta>(c-h)*2/a and tan_theta<pow(c,2)/(2*a*(c-h)):
                print('五边形 1-负')
                record_cal[7] = record_cal[7] + 1
                x = a - pow(2 * a * (c - h) / tan_theta, 0.5)
                mass_center_rect1 = [a - x / 2, b / 2, c / 2]
                m_rect1 = x * c
                mass_center_rect2 = [a / 2 - x / 2, b / 2, (c - (a - x) * tan_theta) /
2]
                m_rect2 = (a - x) * (c - (a - x) * tan_theta)
                mass_center_tria = [(2 * a - 2 * x) / 3, b / 2, c - 2 * (a - x) *
tan_theta / 3]
                m_tria = pow(a - x, 2) * tan_theta / 2
                # print(str(m_rect1 + m_rect2 + m_tria) + ' ' + str(a * h))
                mass_center_sum = [0, 0, 0]
                for i in range(3):
                    mass_center_sum[i] = mass_center_rect1[i] * m_rect1 +
mass_center_rect2[i] * m_rect2 + mass_center_tria[
                        i] * m_tria
                return [k/(m_rect1+m_rect2+m_tria) for k in mass_center_sum]

            # if (h<=c/2 and a*tan_theta/2>=h and 2*tan_theta*a*h>pow(c,2)) or
(h>c/2 and (h+a*tan_theta/2) >= c and tan_theta*2*a*(c-h)>=pow(c,2)): #梯形
            if (h <= c / 2 and tan_theta > pow(c, 2) / (2 * a * h)) or (h > c / 2 and
tan_theta >= pow(c, 2) / (2 * a * (c - h))):
                print('梯形 2-负')
```

```
                    record_cal[8] = record_cal[8] + 1
                    mass_center_rect = [a - a*h/(2*c) + c/(4*tan_theta), b/2, c/2]
                    m_rect = a*h - pow(c,2)/(2*tan_theta)
                    mass_center_tria = [a - a*h/c + c/(6*tan_theta), b/2, c/3]
                    m_tria = pow(c,2)/(2*tan_theta)
                    # print(str(m_rect + m_tria) + ' ' + str(a * h))
                    mass_center_sum = [0, 0, 0]
                    for i in range(3):
                        mass_center_sum[i] = mass_center_rect[i] * m_rect +
mass_center_tria[i] * m_tria
                    return [k/(m_rect+m_tria) for k in mass_center_sum]

                # return [a / 2, b / 2, h / 2]




    def calMassCenters(weight_vs, theta):
        massCenters = []
        for i in range(6):
            weight_v = weight_vs[i+1] #kg/s 质量变化速度
            delta_m = weight_v * 1
            m_current = box_ms[i] - delta_m
            box_ms[i] = m_current
            v_current = m_current/ru
            box_vs[i] = v_current
            box_s = v_current/box_abcs[i][1] #当前的油体侧面积
            h = box_s / box_abcs[i][0] #如果 theta=0，计算出的油面高度
            print('h:'+str(h))
            print('a:' + str(box_abcs[i][0]))
            print('c:'+str(box_abcs[i][2]))
            massCenter = calMassCenterOfSingleBox(h, theta, box_abcs[i][0],
box_abcs[i][1], box_abcs[i][2])
            print(massCenter)
```

```
                for p in range(3):
                    massCenter[p] = massCenter[p]+box_xyz0s[i][p] #将坐标系转换到以飞行器为原点的坐标系
                box_mass_centers[i].append(massCenter)
                massCenters.append(massCenter)
        return massCenters




weight_data=xlrd.open_workbook('data/油箱供油曲线.xlsx')
weight_table = weight_data.sheets()[0]
#print(weight_table.row_values(7199)) #7200 条数据
#第零行是表头
# print(weight_data.keys())
# print(weight_data['时间(s)'])
theta_data=xlrd.open_workbook('data/飞行器俯仰角.xlsx')
theta_table = theta_data.sheets()[0] #7200 条数据

# 创建一个 workbook 设置编码
result_book = xlwt.Workbook(encoding = 'utf-8')
# 创建一个 worksheet
result_sheet = result_book.add_sheet('mass_center')
result_sheet.write(0, 0, label='t/s')
result_sheet.write(0, 1, label='x')
result_sheet.write(0, 2, label='y')
result_sheet.write(0, 3, label='z')
# 写入 excel
# 参数对应 行, 列, 值
```

```
    # print(weight_table.nrows)
    # planet_mass_centers = []
    for i in range(weight_table.nrows-1):
        weight_vs = weight_table.row_values(i+1)
        weight_vs[2] = weight_vs[2] - weight_vs[1] #  油箱 1 输送给油箱 2
        weight_vs[5] = weight_vs[5] - weight_vs[6] #  油箱 6 输送给油箱 5
        theta = theta_table.row_values(i+1)[1]
        print('index:'+str(i+1))
        mass_centers = calMassCenters(weight_vs, theta)
        # print(box_ms)
        sum_m = 3000    #还要加飞行器的质心
        sum_mass_center = [0,0,0] #*sum_m
        for k in range(6):
            mass_center = mass_centers[k]
            # print(mass_center)
            m = box_ms[k]
            sum_m = sum_m + m
            sum_mass_center = [d+m*n for d,n in
zip(sum_mass_center,mass_center)]
        mass_center_planet = [t/sum_m for t in sum_mass_center]
        result_sheet.write(i + 1, 0, label=str(i+1))
        result_sheet.write(i + 1, 1, label=str(mass_center_planet[0]))
        result_sheet.write(i + 1, 2, label=str(mass_center_planet[1]))
        result_sheet.write(i + 1, 3, label=str(mass_center_planet[2]))
    # print(record_cal)
    #  保存
    result_book.save('mass_center_result.xls')



    for p in range(6):
        box_mass_center = box_mass_centers[p]
        result_book = xlwt.Workbook(encoding='utf-8')
        #  创建一个 worksheet
        result_sheet = result_book.add_sheet('mass_center')
```

```
            result_sheet.write(0, 0, label='t/s')
            result_sheet.write(0, 1, label='x')
            result_sheet.write(0, 2, label='y')
            result_sheet.write(0, 3, label='z')
            for i in range(len(box_mass_center)-1):
                mass_center = box_mass_center[i]
                result_sheet.write(i + 1, 0, label=str(i + 1))
                result_sheet.write(i + 1, 1, label=str(mass_center[0]))
                result_sheet.write(i + 1, 2, label=str(mass_center[1]))
                result_sheet.write(i + 1, 3, label=str(mass_center[2]))
            result_book.save('mass_center_box_'+str(p)+'.xls')
```

10.2 问题二算法求解代码

```
import pandas as pd

df_plane_v = pd.read_excel('附件 3-问题 2 数据.xlsx', sheet_name=0)
df_plane_c_gt = pd.read_excel('附件 3-问题 2 数据.xlsx', sheet_name=1)
plane_v = df_plane_v['耗油速度(kg/s)'].tolist()
# while plane_v[0] == 0:
#         plane_v = plane_v[1:]
plane_c_gt_x = df_plane_c_gt['X 坐标（米）']
plane_c_gt_y = df_plane_c_gt['y 坐标（米）']
plane_c_gt_z = df_plane_c_gt['z 坐标（米）']
plane_c_gt = [(plane_c_gt_x[i], plane_c_gt_y[i], plane_c_gt_z[i]) for i in
range(len(plane_c_gt_x))]
total_time = 7200
sum_l = 0
beiyong_per = 0.700
favor = 0.09
good_percentages = []
for split_time in range(60, 90):
    # if total_time % split_time == 0:
    percentages = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
    final_max_min_dis = None
    choices = [[2], [3], [4], [5],
               [2, 3], [2, 4], [2, 5], [3, 4], [3, 5], [4, 5], [1, 2], [1, 3], [1, 4], [1, 5], [6, 2], [6,
3],
               [6, 4],
               [6, 5], [1, 2, 6], [1, 3, 6], [1, 4, 6], [1, 5, 6], [1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3,
4],
```

```
                    [1, 3, 5],
                    [1, 4, 5],
                    [6, 2, 3], [6, 2, 4], [6, 2, 5], [6, 3, 4], [6, 3, 5], [6, 4, 5]]
    upper_bounds = [1.1, 1.8, 1.7, 1.5, 1.6, 1.1]
    box_centers = [(8.91304348, 1.20652174, 0.61669004),
                   (6.91304348, -1.39347826, 0.21669004),
                   (-1.68695652, 1.20652174, -0.28330996),
                   (3.11304348, 0.60652174, -0.18330996),
                   (-5.28695652, -0.29347826, 0.41669004),
                   (-2.08695652, -1.49347826, 0.21669004)]
    box_sizes = [(1.5, 0.9, 0.3), (2.2, 0.8, 1.1), (2.4, 1.1, 0.9), (1.7, 1.3, 1.2), (2.4, 1.2, 1), (2.4,
1, 0.5)]
    box_max_volume = [344.25,
                      1645.6,
                      2019.6,
                      2254.2,
                      2448,
                      1020]
    last_oil = [255,
                1275,
                1785,
                1615,
                2210,
                680]
    best_oil = None
    kexin_flag = True
    begin_time = 0
    while begin_time + 60 <= 7200:
        final_choice = None
        min_dis = None
        oil_map = {}
        v_best = []
        best_dist = []
        end_time = min(begin_time + split_time, 7200)
        # print(begin_time)
        need_oil = sum(plane_v[begin_time: end_time])
        for choice in choices:
            oil = [item for item in last_oil]  # 为每个可能性初始化
            v = []
            max_dis = None  # 这个选择产生的 60s 里的距离最大值
            choice_tmp_dist = []
            if len(choice) == 1:
                v = []
                # 一个油箱
```

```
                box_i = choice[0] - 1
                # 先判断符不符合
                if upper_bounds[box_i] < max(plane_v[begin_time:end_time]) or \
                        oil[
                            box_i] < sum(plane_v[begin_time:end_time]):
                    continue
                else:
                    for i in range(begin_time, end_time):
                        v.append([plane_v[i]])
                        oil[box_i] -= plane_v[i]
                        c = [0, 0, 0]
                        for vec in [(m * box_center[0], m * box_center[1],
                                     m * (box_center[2] + (m / (
                                             850 * box_size[0] * box_size[1]) -
box_size[2]) / 2))
                                    for (box_center, m, box_size) in zip(box_centers,
oil, box_sizes)]:
                            c[0] += (vec[0] / (3000 + sum(oil)))
                            c[1] += (vec[1] / (3000 + sum(oil)))
                            c[2] += (vec[2] / (3000 + sum(oil)))
                        c_gt = plane_c_gt[i]
                        dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2]
- c[2], 2)) ** 0.5
                        if max_dis is None:
                            max_dis = dis
                        else:
                            max_dis = max(max_dis, dis)
                        choice_tmp_dist.append("%d, %s" % (i, max_dis))
            elif len(choice) == 2:
                if choice == [1, 2] or choice == [2, 1]:
                    v = []
                    choice = [1, 2]
                    box_i_a = 1 - 1
                    box_i_b = 2 - 1
                    if upper_bounds[box_i_b] < max(
                            plane_v[begin_time:end_time]) or oil[
                        box_i_b] < sum(plane_v[begin_time:end_time]):
                        continue
                    for i in range(begin_time, end_time):
                        v_b = plane_v[i]
                        v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                        if oil[box_i_a] - v_a < 0:
                            # 没油了
                            v_a = 0
```

```
                                oil[box_i_a] -= v_a
                                oil[box_i_b] -= (v_b - v_a)
                                v.append([v_a, v_b])
                                c = [0, 0, 0]
                                for vec in [(m * box_center[0], m * box_center[1],
                                             m * (box_center[2] + (m / (
                                                     850 * box_size[0] * box_size[1]) -
box_size[2]) / 2))
                                            for (box_center, m, box_size) in zip(box_centers,
oil, box_sizes)]:
                                    c[0] += (vec[0] / (3000 + sum(oil)))
                                    c[1] += (vec[1] / (3000 + sum(oil)))
                                    c[2] += (vec[2] / (3000 + sum(oil)))
                                c_gt = plane_c_gt[i]
                                dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2]
- c[2], 2)) ** 0.5
                                if max_dis is None:
                                    max_dis = dis
                                else:
                                    if dis > max_dis:
                                        max_dis = dis
                                choice_tmp_dist.append("%d, %s" % (i, max_dis))
                        elif choice == [5, 6] or choice == [6, 5]:
                            choice = [6, 5]
                            v = []
                            box_i_a = 6 - 1
                            box_i_b = 5 - 1
                            # print(upper_bounds[box_i_b])
                            # print(max(plane_v[split_time * split_iter: split_time * (split_iter +
1)]))
                            if upper_bounds[box_i_b] < max(
                                    plane_v[begin_time:end_time]) or oil[
                                box_i_b] < sum(plane_v[begin_time:end_time]):
                                continue
                            for i in range(begin_time, end_time):
                                v_b = plane_v[i]
                                v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                if oil[box_i_a] - v_a < 0:
                                    # 没油了
                                    v_a = 0
                                v.append([v_a, v_b])
                                oil[box_i_a] -= v_a
                                oil[box_i_b] -= (v_b - v_a)
                                c = [0, 0, 0]
```

```
                        for vec in [(m * box_center[0], m * box_center[1],
                                     m * (box_center[2] + (m / (
                                             850 * box_size[0] * box_size[1]) -
box_size[2]) / 2))
                                    for (box_center, m, box_size) in zip(box_centers,
oil, box_sizes)]:
                            c[0] += (vec[0] / (3000 + sum(oil)))
                            c[1] += (vec[1] / (3000 + sum(oil)))
                            c[2] += (vec[2] / (3000 + sum(oil)))
                        c_gt = plane_c_gt[i]
                        dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2]
- c[2], 2)) ** 0.5
                        if max_dis is None or dis > max_dis:
                            max_dis = dis
                        choice_tmp_dist.append("%d, %s" % (i, max_dis))
                elif 6 not in choice and 1 not in choice:
                    box_i_a = choice[0] - 1
                    box_i_b = choice[1] - 1
                    best_max_dis = None  # percentage 去刷新，取最大值最小的一种
方案
                    best_c_oil = None  # 保存最大值最小的方案的油量
                    v_tmp_best = []
                    good_percentage = None
                    dist_best = []
                    for percentage in percentages:
                        v_tmp = []
                        dist_tmp = []
                        tmp_max_dis = None
                        c_oil = [item for item in oil]
                        flag = True
                        for i in range(begin_time, end_time):
                            v_a = percentage * plane_v[i]
                            v_b = (1 - percentage) * plane_v[i]
                            v_tmp.append([v_a, v_b])
                            if upper_bounds[box_i_a] < v_a or upper_bounds[box_i_b] <
v_b:
                                flag = False
                                break
                            elif c_oil[box_i_a] - v_a < 0 or c_oil[box_i_b] - v_b < 0:
                                flag = False
                                break
                            else:
                                c_oil[box_i_a] -= v_a
                                c_oil[box_i_b] -= v_b
```

```
                                    c = [0, 0, 0]
                                    for vec in [(m * box_center[0], m * box_center[1],
                                                 m * (box_center[2] + (m / (
                                                         850 * box_size[0] *
box_size[1]) - box_size[2]) / 2))
                                                for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                        c[0] += (vec[0] / (3000 + sum(c_oil)))
                                        c[1] += (vec[1] / (3000 + sum(c_oil)))
                                        c[2] += (vec[2] / (3000 + sum(c_oil)))
                                    c_gt = plane_c_gt[i]
                                    dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2],

2)) ** 0.5
                                    if tmp_max_dis is None:
                                        tmp_max_dis = dis
                                    else:
                                        if dis > tmp_max_dis:
                                            tmp_max_dis = dis
                                    dist_tmp.append("%d, %s" % (i, tmp_max_dis))

                            if flag:
                                if best_max_dis is None or (tmp_max_dis is not None and
tmp_max_dis < best_max_dis):
                                    best_max_dis = tmp_max_dis
                                    best_c_oil = c_oil[:]
                                    v_tmp_best = v_tmp
                                    good_percentage = percentage
                                    dist_best = dist_tmp
                        if best_c_oil is not None:
                            oil = best_c_oil[:]
                            max_dis = best_max_dis
                            v = v_tmp_best
                            good_percentages.append(good_percentage)
                            choice_tmp_dist = dist_best
                elif len(choice) == 3:
                    if 1 in choice and 2 in choice and 6 not in choice:
                        choice.sort()
                        box_i_a = 1 - 1
                        box_i_b = 2 - 1
                        box_i_c = choice[-1] - 1
                        c_oil = [item for item in oil]
                        best_c_oil = None
```

```
                best_max_dis = None  # percentage 去刷新，取最大值最小的一种
方案
                v_tmp_best = []
                good_percentage = None
                dist_best = []
                for percentage in percentages:
                    tmp_max_dis = None
                    c_oil = [item for item in oil]
                    flag = True
                    v_tmp = []
                    dist_tmp = []
                    for i in range(begin_time, end_time):
                        v_b = percentage * plane_v[i]
                        v_c = (1 - percentage) * plane_v[i]
                        v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                        v_tmp.append([v_a, v_b, v_c])
                        if upper_bounds[box_i_b] < v_b or upper_bounds[box_i_c] <
v_c:
                            flag = False
                            break
                        elif c_oil[box_i_b] - v_b < 0 or c_oil[box_i_c] - v_c < 0:
                            flag = False
                            break
                        else:
                            if c_oil[box_i_a] - v_a < 0:
                                v_a = 0
                            c_oil[box_i_b] -= (v_b - v_a)
                            c_oil[box_i_c] -= v_c
                            c_oil[box_i_a] -= v_a
                            c = [0, 0, 0]
                            for vec in [(m * box_center[0], m * box_center[1],
                                         m * (box_center[2] + (m / (
                                                 850  *  box_size[0]  *
box_size[1]) - box_size[2]) / 2))
                                        for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                c[0] += (vec[0] / (3000 + sum(c_oil)))
                                c[1] += (vec[1] / (3000 + sum(c_oil)))
                                c[2] += (vec[2] / (3000 + sum(c_oil)))
                            c_gt = plane_c_gt[i]
                            dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2],

2)) ** 0.5
```

```
                                    if tmp_max_dis is None or dis > tmp_max_dis:
                                        tmp_max_dis = dis
                                    dist_tmp.append("%d, %s" % (i, tmp_max_dis))
                            if flag:
                                if best_max_dis is None or (tmp_max_dis is not None and
tmp_max_dis < best_max_dis):
                                    best_max_dis = tmp_max_dis
                                    best_c_oil = c_oil[:]
                                    v_tmp_best = v_tmp
                                    good_percentage = percentage
                                    dist_best = dist_tmp
                        if best_c_oil is not None:
                            oil = best_c_oil
                            max_dis = best_max_dis
                            v = v_tmp_best
                            good_percentages.append(good_percentage)
                            choice_tmp_dist = dist_best
                    elif 5 in choice and 6 in choice and 1 not in choice:
                        box_i_a = 6 - 1
                        box_i_b = 5 - 1
                        choice.sort()
                        box_i_c = choice[0] - 1
                        c_oil = [item for item in oil]
                        best_c_oil = None
                        best_max_dis = None  # percentage 去刷新，取最大值最小的一种
方案
                        v_tmp_best = []
                        good_percentage = None
                        dist_best = None
                        for percentage in percentages:
                            tmp_max_dis = None
                            c_oil = [item for item in oil]
                            flag = True
                            v_tmp = []
                            dist_tmp = []
                            for i in range(begin_time, end_time):
                                v_b = percentage * plane_v[i]
                                v_c = (1 - percentage) * plane_v[i]
                                v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                v_tmp.append([v_c, v_b, v_a])
                                if upper_bounds[box_i_b] < v_b or upper_bounds[box_i_c] <
v_c:
                                    flag = False
                                    break
```

```
                                elif c_oil[box_i_b] - v_b < 0 or c_oil[box_i_c] - v_c < 0:
                                    flag = False
                                    break
                                else:
                                    if c_oil[box_i_a] - v_a < 0:
                                        v_a = 0
                                    c_oil[box_i_b] -= (v_b - v_a)
                                    c_oil[box_i_c] -= v_c
                                    c_oil[box_i_a] -= v_a
                                    c = [0, 0, 0]
                                    for vec in [(m * box_center[0], m * box_center[1],
                                                 m * (box_center[2] + (m / (
                                                         850 * box_size[0] *
box_size[1]) - box_size[2]) / 2))
                                                for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                        c[0] += (vec[0] / (3000 + sum(c_oil)))
                                        c[1] += (vec[1] / (3000 + sum(c_oil)))
                                        c[2] += (vec[2] / (3000 + sum(c_oil)))
                                    c_gt = plane_c_gt[i]
                                    dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2],

2)) ** 0.5
                                    if tmp_max_dis is None or dis > tmp_max_dis:
                                        tmp_max_dis = dis
                                    dist_tmp.append("%d, %s" % (i, tmp_max_dis))
                            if flag:
                                if best_max_dis is None or (tmp_max_dis is not None and
tmp_max_dis < best_max_dis):
                                    best_max_dis = tmp_max_dis
                                    best_c_oil = c_oil[:]
                                    v_tmp_best = v_tmp
                                    dist_best = dist_tmp
                                    good_percentage = percentage
                        if best_c_oil is not None:
                            oil = best_c_oil
                            max_dis = best_max_dis
                            v = v_tmp_best
                            good_percentages.append(good_percentage)
                            choice_tmp_dist = dist_best
```

```
            # oil_map[tuple(choice)] = oil
            if max_dis is None:
                continue
            if min_dis is None or min_dis > max_dis:
                min_dis = max_dis
                final_choice = choice
                best_oil = [item for item in oil]
                v_best = v
                best_dist = choice_tmp_dist
        last_oil_total = sum(last_oil)
        last_oil = [item for item in best_oil]
        # pprint.pprint(oil_map)
        # print(final_choice)

        # print(min_dis)
        favor_flag = True
        for l in range(split_time):
            if max([float(best_dist[p].split(',')[1]) for p in range(len(best_dist))]) <= favor:
                favor_flag = False
                l = split_time
                break
            if l < 60:
                if float(best_dist[l].split(',')[1]) > favor:
                    favor_flag = False
                    l = split_time
                    break
            else:
                if float(best_dist[l].split(',')[1]) > favor:
                    begin_time = begin_time + l
                    break
        if not favor_flag:
            if final_choice is None:
                kexin_flag = False
                break
            if final_max_min_dis is None:
                final_max_min_dis = min_dis
            else:
                if min_dis > final_max_min_dis:
                    final_max_min_dis = max(final_max_min_dis, min_dis)
            # for d in best_dist:
            #     print(d)
            begin_time += split_time
            # print(v_best)
```

```
            else:
                if final_choice is None:
                    kexin_flag = False
                    break
                if final_max_min_dis is None or min_dis > final_max_min_dis:
                    final_max_min_dis = max([float(best_dist[p].split(',')[1]) for p in range(l)])
                # for d in best_dist[:l]:
                #     print(d)
            sum_l += l
            v_best = v_best[:l]
            # print(v_best)

            # print(need_oil)
            # print(last_oil)
            # print(abs(last_oil_total - (need_oil + sum(last_oil))) < 0.01)
            # print()
        # print("==========")
        # print(sum_l)
        print(split_time)
        if kexin_flag == False:
            print("不行")
        else:
            print("%f" % final_max_min_dis)
```

10.3 问题三算法求解代码

```
import pandas as pd

df_plane_v = pd.read_excel('附件 4-问题 3 数据.xlsx', sheet_name=0)
df_plane_c_gt = pd.read_excel('附件 4-问题 3 数据.xlsx', sheet_name=1)
plane_v = df_plane_v['耗油速度(kg/s)'].tolist()
# while plane_v[0] == 0:
#     plane_v = plane_v[1:]
plane_c_gt_x = df_plane_c_gt['X 坐标（米）']
plane_c_gt_y = df_plane_c_gt['y 坐标（米）']
plane_c_gt_z = df_plane_c_gt['z 坐标（米）']
plane_c_gt = [(plane_c_gt_x[i], plane_c_gt_y[i], plane_c_gt_z[i]) for i in
range(len(plane_c_gt_x))]
total_time = 7200

good_percentages = []
beiyong_per = 0.5
for split_time in range(60, 61):
    if total_time % split_time == 0:
        percentages = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

```
        final_max_min_dis = None
        choices = [[2], [3], [4], [5],
                   [2, 3], [2, 4], [2, 5], [3, 4], [3, 5], [4, 5], [1, 2], [1, 3], [1, 4], [1, 5], [6, 2], [6, 3],
                   [6, 4],
                   [6, 5], [1, 2, 6], [1, 3, 6], [1, 4, 6], [1, 5, 6], [1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4],
                   [1, 3, 5],
                   [1, 4, 5],
                   [6, 2, 3], [6, 2, 4], [6, 2, 5], [6, 3, 4], [6, 3, 5], [6, 4, 5]]
        upper_bounds = [1.1, 1.8, 1.7, 1.5, 1.6, 1.1]
        box_centers = [(8.91304348, 1.20652174, 0.61669004),
                       (6.91304348, -1.39347826, 0.21669004),
                       (-1.68695652, 1.20652174, -0.28330996),
                       (3.11304348, 0.60652174, -0.18330996),
                       (-5.28695652, -0.29347826, 0.41669004),
                       (-2.08695652, -1.49347826, 0.21669004)]
        box_sizes = [(1.5, 0.9, 0.3), (2.2, 0.8, 1.1), (2.4, 1.1, 0.9), (1.7, 1.3, 1.2), (2.4, 1.2, 1), (2.4, 1, 0.5)]

        # last_oil = [344, 1645, 1580.63568964, 1399.56408643, 2432.81261397, 252.98760996]
        # last_oil = [344.25, 1645.6, 1632.4227398755725, 1349.0153755156696, 2357.77614575568, 326.11040885307904]
        # last_oil = [344.25, 1645.6, 1746.176871030044, 1761.6501267521792, 2419.770867096817, 587.7268051209606]
        # last_oil = [344.25, 1645.6, 1857.8788934335935, 1929.3795356749902, 2394.915901490579, 758.1503394008387]
        last_oil =[300, 1645, 1613.0288566563802, 1684.081124389051, 2434.6588693662766, 403.4058195882918]
        best_oil = None
        kexin_flag = True
        for split_iter in range(0, int(7200 / split_time)):
            final_choice = None
            min_dis = None
            oil_map = {}
            v_best = []

            need_oil = sum(plane_v[split_time * split_iter: split_time * (split_iter + 1)])
            for choice in choices:
                oil = [item for item in last_oil]  # 为每个可能性初始化
                v = []
                max_dis = None  # 这个选择产生的 60s 里的距离最大值
                if len(choice) == 1:
```

```
                        v = []
                        # 一个油箱
                        box_i = choice[0] - 1
                        # 先判断符不符合
                        if upper_bounds[box_i] < max(plane_v[split_time * split_iter:
split_time * (split_iter + 1)]) or \
                                oil[
                                    box_i] < sum(plane_v[split_time * split_iter: split_time
* (split_iter + 1)]):
                            continue
                        else:
                            for i in range(split_time * split_iter, split_time * (split_iter + 1)):
                                v.append([plane_v[i]])
                                oil[box_i] -= plane_v[i]
                                c = [0, 0, 0]
                                for vec in [(m * box_center[0], m * box_center[1],
                                             m * (box_center[2] + (m / (
                                                     850 * box_size[0] * box_size[1]) -
box_size[2]) / 2))
                                            for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                    c[0] += (vec[0] / (3000 + sum(oil)))
                                    c[1] += (vec[1] / (3000 + sum(oil)))
                                    c[2] += (vec[2] / (3000 + sum(oil)))
                                c_gt = plane_c_gt[i]
                                dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2], 2)) ** 0.5
                                if max_dis is None:
                                    max_dis = dis
                                else:
                                    max_dis = max(max_dis, dis)
                    elif len(choice) == 2:
                        if choice == [1, 2] or choice == [2, 1]:
                            v = []
                            choice = [1, 2]
                            box_i_a = 1 - 1
                            box_i_b = 2 - 1
                            if upper_bounds[box_i_b] < max(
                                    plane_v[split_time * split_iter: split_time * (split_iter +
1)]) or oil[
                                box_i_b] < sum(plane_v[split_time * split_iter: split_time *
(split_iter + 1)]):
                                continue
                            for i in range(split_time * split_iter, split_time * (split_iter + 1)):
```

```
                                v_b = plane_v[i]
                                v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                if oil[box_i_a] - v_a < 0:
                                    # 没油了
                                    v_a = 0
                                oil[box_i_a] -= v_a
                                oil[box_i_b] -= (v_b - v_a)
                                v.append([v_a, v_b])
                                c = [0, 0, 0]
                                for vec in [(m * box_center[0], m * box_center[1],
                                             m * (box_center[2] + (m / (
                                                     850 * box_size[0] * box_size[1]) -
box_size[2]) / 2))
                                            for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                    c[0] += (vec[0] / (3000 + sum(oil)))
                                    c[1] += (vec[1] / (3000 + sum(oil)))
                                    c[2] += (vec[2] / (3000 + sum(oil)))
                                c_gt = plane_c_gt[i]
                                dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2], 2)) ** 0.5
                                if max_dis is None:
                                    max_dis = dis
                                else:
                                    if dis > max_dis:
                                        max_dis = dis
                        elif choice == [5, 6] or choice == [6, 5]:
                            choice = [6, 5]
                            v = []
                            box_i_a = 6 - 1
                            box_i_b = 5 - 1
                            # print(upper_bounds[box_i_b])
                            # print(max(plane_v[split_time * split_iter: split_time * (split_iter
+ 1)]))
                            if upper_bounds[box_i_b] < max(
                                    plane_v[split_time * split_iter: split_time * (split_iter +
1)]) or oil[
                                box_i_b] < sum(plane_v[split_time * split_iter: split_time *
(split_iter + 1)]):
                                continue
                            for i in range(split_time * split_iter, split_time * (split_iter + 1)):
                                v_b = plane_v[i]
                                v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                if oil[box_i_a] - v_a < 0:
```

```
                                    # 没油了
                                    v_a = 0
                                v.append([v_a, v_b])
                                oil[box_i_a] -= v_a
                                oil[box_i_b] -= (v_b - v_a)
                                c = [0, 0, 0]
                                for vec in [(m * box_center[0], m * box_center[1],
                                             m * (box_center[2] + (m / (
                                                     850 * box_size[0] * box_size[1]) -
box_size[2]) / 2))
                                            for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                    c[0] += (vec[0] / (3000 + sum(oil)))
                                    c[1] += (vec[1] / (3000 + sum(oil)))
                                    c[2] += (vec[2] / (3000 + sum(oil)))
                                c_gt = plane_c_gt[i]
                                dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2], 2)) ** 0.5
                                if max_dis is None or dis > max_dis:
                                    max_dis = dis
                        elif 6 not in choice and 1 not in choice:
                            box_i_a = choice[0] - 1
                            box_i_b = choice[1] - 1
                            best_max_dis = None  # percentage 去刷新，取最大值最小的
一种方案
                            best_c_oil = None  # 保存最大值最小的方案的油量
                            v_tmp_best = []
                            good_percentage = None
                            for percentage in percentages:
                                v_tmp = []
                                tmp_max_dis = None
                                c_oil = [item for item in oil]
                                flag = True
                                for i in range(split_time * split_iter, split_time * (split_iter +
1)):
                                    v_a = percentage * plane_v[i]
                                    v_b = (1 - percentage) * plane_v[i]
                                    v_tmp.append([v_a, v_b])
                                    if upper_bounds[box_i_a] < v_a or
upper_bounds[box_i_b] < v_b:
                                        flag = False
                                        break
                                    elif c_oil[box_i_a] - v_a < 0 or c_oil[box_i_b] - v_b < 0:
                                        flag = False
```

```
                                        break
                                    else:
                                        c_oil[box_i_a] -= v_a
                                        c_oil[box_i_b] -= v_b
                                        c = [0, 0, 0]
                                        for vec in [(m * box_center[0], m * box_center[1],
                                                     m * (box_center[2] + (m / (
                                                             850 * box_size[0] *
box_size[1]) - box_size[2]) / 2))
                                                    for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                            c[0] += (vec[0] / (3000 + sum(c_oil)))
                                            c[1] += (vec[1] / (3000 + sum(c_oil)))
                                            c[2] += (vec[2] / (3000 + sum(c_oil)))
                                        c_gt = plane_c_gt[i]
                                        dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1],
2) + pow(c_gt[2] - c[2],

2)) ** 0.5
                                        if tmp_max_dis is None:
                                            tmp_max_dis = dis
                                        else:
                                            if dis > tmp_max_dis:
                                                tmp_max_dis = dis
                                if flag:
                                    if best_max_dis is None or (tmp_max_dis is not None
and tmp_max_dis < best_max_dis):
                                        best_max_dis = tmp_max_dis
                                        best_c_oil = c_oil[:]
                                        v_tmp_best = v_tmp
                                        good_percentage = percentage
                            if best_c_oil is not None:
                                oil = best_c_oil[:]
                                max_dis = best_max_dis
                                v = v_tmp_best
                                good_percentages.append(good_percentage)
                        elif len(choice) == 3:
                            if 1 in choice and 2 in choice and 6 not in choice:
                                choice.sort()
                                box_i_a = 1 - 1
                                box_i_b = 2 - 1
                                box_i_c = choice[-1] - 1
                                c_oil = [item for item in oil]
                                best_c_oil = None
```

```
                best_max_dis = None  # percentage 去刷新，取最大值最小的一种方案
                v_tmp_best = []
                good_percentage = None
                for percentage in percentages:
                    tmp_max_dis = None
                    c_oil = [item for item in oil]
                    flag = True
                    v_tmp = []
                    for i in range(split_time * split_iter, split_time * (split_iter + 1)):
                        v_b = percentage * plane_v[i]
                        v_c = (1 - percentage) * plane_v[i]
                        v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                        v_tmp.append([v_a, v_b, v_c])
                        if upper_bounds[box_i_b] < v_b or upper_bounds[box_i_c] < v_c:
                            flag = False
                            break
                        elif c_oil[box_i_b] - v_b < 0 or c_oil[box_i_c] - v_c < 0:
                            flag = False
                            break
                        else:
                            if c_oil[box_i_a] - v_a < 0:
                                v_a = 0
                            c_oil[box_i_b] -= (v_b - v_a)
                            c_oil[box_i_c] -= v_c
                            c_oil[box_i_a] -= v_a
                            c = [0, 0, 0]
                            for vec in [(m * box_center[0], m * box_center[1],
                                         m * (box_center[2] + (m / (
                                                 850 * box_size[0] * box_size[1]) - box_size[2]) / 2))
                                        for (box_center, m, box_size) in zip(box_centers, oil, box_sizes)]:
                                c[0] += (vec[0] / (3000 + sum(c_oil)))
                                c[1] += (vec[1] / (3000 + sum(c_oil)))
                                c[2] += (vec[2] / (3000 + sum(c_oil)))
                            c_gt = plane_c_gt[i]
                            dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2] - c[2],

                            2)) ** 0.5
                            if tmp_max_dis is None or dis > tmp_max_dis:
```

```
                                        tmp_max_dis = dis
                            if flag:
                                if best_max_dis is None or (tmp_max_dis is not None
and tmp_max_dis < best_max_dis):
                                    best_max_dis = tmp_max_dis
                                    best_c_oil = c_oil[:]
                                    v_tmp_best = v_tmp
                                    good_percentage = percentage
                        if best_c_oil is not None:
                            oil = best_c_oil
                            max_dis = best_max_dis
                            v = v_tmp_best
                            good_percentages.append(good_percentage)
                    elif 5 in choice and 6 in choice and 1 not in choice:
                        box_i_a = 6 - 1
                        box_i_b = 5 - 1
                        choice.sort()
                        box_i_c = choice[0] - 1
                        c_oil = [item for item in oil]
                        best_c_oil = None
                        best_max_dis = None  # percentage 去刷新，取最大值最小的
一种方案
                        v_tmp_best = []
                        good_percentage = None
                        for percentage in percentages:
                            tmp_max_dis = None
                            c_oil = [item for item in oil]
                            flag = True
                            v_tmp = []
                            for i in range(split_time * split_iter, split_time * (split_iter +
1)):
                                v_b = percentage * plane_v[i]
                                v_c = (1 - percentage) * plane_v[i]
                                v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                v_tmp.append([v_c, v_b, v_a])
                                if upper_bounds[box_i_b] < v_b or
upper_bounds[box_i_c] < v_c:
                                    flag = False
                                    break
                                elif c_oil[box_i_b] - v_b < 0 or c_oil[box_i_c] - v_c < 0:
                                    flag = False
                                    break
                                else:
                                    if c_oil[box_i_a] - v_a < 0:
```

```
                                        v_a = 0
                                    c_oil[box_i_b] -= (v_b - v_a)
                                    c_oil[box_i_c] -= v_c
                                    c_oil[box_i_a] -= v_a
                                    c = [0, 0, 0]
                                    for vec in [(m * box_center[0], m * box_center[1],
                                                 m * (box_center[2] + (m / (
                                                         850 * box_size[0] *
box_size[1]) - box_size[2]) / 2))
                                                for (box_center, m, box_size) in
zip(box_centers, oil, box_sizes)]:
                                        c[0] += (vec[0] / (3000 + sum(c_oil)))
                                        c[1] += (vec[1] / (3000 + sum(c_oil)))
                                        c[2] += (vec[2] / (3000 + sum(c_oil)))
                                    c_gt = plane_c_gt[i]
                                    dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1],
2) + pow(c_gt[2] - c[2],

2)) ** 0.5
                                    if tmp_max_dis is None or dis > tmp_max_dis:
                                        tmp_max_dis = dis
                                if flag:
                                    if best_max_dis is None or (tmp_max_dis is not None
and tmp_max_dis < best_max_dis):
                                        best_max_dis = tmp_max_dis
                                        best_c_oil = c_oil[:]
                                        v_tmp_best = v_tmp
                                        good_percentage = percentage
                            if best_c_oil is not None:
                                oil = best_c_oil
                                max_dis = best_max_dis
                                v = v_tmp_best
                                good_percentages.append(good_percentage)
                    # oil_map[tuple(choice)] = oil
                    if max_dis is None:
                        continue
                    if min_dis is None or min_dis > max_dis:
                        min_dis = max_dis
                        final_choice = choice
                        best_oil = [item for item in oil]
                        v_best = v
                last_oil_total = sum(last_oil)
                last_oil = [item for item in best_oil]
                # pprint.pprint(oil_map)
```

```
                print(final_choice)
                print(v_best)
                # print(min_dis)
                if final_choice is None:
                    kexin_flag = False
                    break
                if final_max_min_dis is None:
                    final_max_min_dis = min_dis
                else:
                    if min_dis > final_max_min_dis:
                        final_max_min_dis = max(final_max_min_dis, min_dis)
                # print(need_oil)
                # print(last_oil)
                # print(abs(last_oil_total - (need_oil + sum(last_oil))) < 0.01)
                # print()
            print("==========")
            print(split_time)
            if kexin_flag == False:
                print("不行")
            else:
                print("质心距离最大值：%f" % final_max_min_dis)
```

10.4 问题四算法求解代码

```
import pandas as pd
from mass_center_cal_interface import calMassCenters

df_plane_v = pd.read_excel('附件 5-问题 4 数据.xlsx', sheet_name=0)
df_plane_theta = pd.read_excel('附件 5-问题 4 数据.xlsx', sheet_name=1)
plane_v = df_plane_v['耗油速度(kg/s)'].tolist()
plane_theta = df_plane_theta['俯仰角(度)'].tolist()
total_time = 7200
beiyong_per = 0.800
favor = 1.0
good_percentages = []
for split_time in range(60, 90):
    # if total_time % split_time == 0:
    percentages = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
    final_max_min_dis = None
    choices = [[2], [3], [4], [5],
               [2, 3], [2, 4], [2, 5], [3, 4], [3, 5], [4, 5], [1, 2], [1, 3], [1, 4], [1, 5], [6, 2], [6,
3],
               [6, 4],
```

```
                    [6, 5], [1, 2, 6], [1, 3, 6], [1, 4, 6], [1, 5, 6], [1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3,
4],
                    [1, 3, 5],
                    [1, 4, 5],
                    [6, 2, 3], [6, 2, 4], [6, 2, 5], [6, 3, 4], [6, 3, 5], [6, 4, 5]]
    upper_bounds = [1.1, 1.8, 1.7, 1.5, 1.6, 1.1]
    box_centers = [(8.91304348, 1.20652174, 0.61669004),
                   (6.91304348, -1.39347826, 0.21669004),
                   (-1.68695652, 1.20652174, -0.28330996),
                   (3.11304348, 0.60652174, -0.18330996),
                   (-5.28695652, -0.29347826, 0.41669004),
                   (-2.08695652, -1.49347826, 0.21669004)]
    box_sizes = [(1.5, 0.9, 0.3), (2.2, 0.8, 1.1), (2.4, 1.1, 0.9), (1.7, 1.3, 1.2), (2.4, 1.2, 1), (2.4,
1, 0.5)]
    box_max_volume = [344.25,
                      1645.6,
                      2019.6,
                      2254.2,
                      2448,
                      1020]
    last_oil = [255,
                1275,
                1785,
                1615,
                2210,
                680]
    best_oil = None
    kexin_flag = True
    begin_time = 0
    while begin_time + 60 <= 7200:
        final_choice = None
        min_dis = None
        oil_map = {}
        v_best = []
        best_dist = []
        # print(begin_time)
        end_time = min(begin_time + split_time, 7200)

        need_oil = sum(plane_v[begin_time: end_time])
        for choice in choices:
            oil = [item for item in last_oil]  # 为每个可能性初始化
            v = []
            max_dis = None  # 这个选择产生的 60s 里的距离最大值
            choice_tmp_dist = []
```

```
                if len(choice) == 1:
                    v = []
                    # 一个油箱
                    box_i = choice[0] - 1
                    # 先判断符不符合
                    if upper_bounds[box_i] < max(plane_v[begin_time:end_time]) or \
                            oil[
                                box_i] < sum(plane_v[begin_time:end_time]):
                        continue
                    else:
                        for i in range(begin_time, end_time):
                            v.append([plane_v[i]])
                            oil[box_i] -= plane_v[i]
                            c = calMassCenters(oil, theta=plane_theta[i])
                            c_gt = [0, 0, 0]
                            dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2]
- c[2], 2)) ** 0.5
                            if max_dis is None:
                                max_dis = dis
                            else:
                                max_dis = max(max_dis, dis)
                            choice_tmp_dist.append("%d, %s" % (i, max_dis))
                elif len(choice) == 2:
                    if choice == [1, 2] or choice == [2, 1]:
                        v = []
                        choice = [1, 2]
                        box_i_a = 1 - 1
                        box_i_b = 2 - 1
                        if upper_bounds[box_i_b] < max(
                                plane_v[begin_time:end_time]) or oil[
                            box_i_b] < sum(plane_v[begin_time:end_time]):
                            continue
                        for i in range(begin_time, end_time):
                            v_b = plane_v[i]
                            v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                            if oil[box_i_a] - v_a < 0:
                                # 没油了
                                v_a = 0
                            oil[box_i_a] -= v_a
                            oil[box_i_b] -= (v_b - v_a)
                            v.append([v_a, v_b])
                            c = calMassCenters(oil, theta=plane_theta[i])
                            c_gt = [0, 0, 0]
                            dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2]
```

```
- c[2], 2)) ** 0.5
                            if max_dis is None:
                                max_dis = dis
                            else:
                                if dis > max_dis:
                                    max_dis = dis
                            choice_tmp_dist.append("%d, %s" % (i, max_dis))
                    elif choice == [5, 6] or choice == [6, 5]:
                        choice = [6, 5]
                        v = []
                        box_i_a = 6 - 1
                        box_i_b = 5 - 1
                        # print(upper_bounds[box_i_b])
                        # print(max(plane_v[split_time * split_iter: split_time * (split_iter +
1)]))
                        if upper_bounds[box_i_b] < max(
                                plane_v[begin_time:end_time]) or oil[
                            box_i_b] < sum(plane_v[begin_time:end_time]):
                            continue
                        for i in range(begin_time, end_time):
                            v_b = plane_v[i]
                            v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                            if oil[box_i_a] - v_a < 0:
                                # 没油了
                                v_a = 0
                            v.append([v_a, v_b])
                            oil[box_i_a] -= v_a
                            oil[box_i_b] -= (v_b - v_a)
                            c = calMassCenters(oil, theta=plane_theta[i])
                            c_gt = [0, 0, 0]
                            dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) + pow(c_gt[2]
- c[2], 2)) ** 0.5
                            if max_dis is None or dis > max_dis:
                                max_dis = dis
                            choice_tmp_dist.append("%d, %s" % (i, max_dis))
                    elif 6 not in choice and 1 not in choice:
                        box_i_a = choice[0] - 1
                        box_i_b = choice[1] - 1
                        best_max_dis = None  # percentage 去刷新，取最大值最小的一种
方案
                        best_c_oil = None  # 保存最大值最小的方案的油量
                        v_tmp_best = []
                        good_percentage = None
                        dist_best = []
```

```
                for percentage in percentages:
                    v_tmp = []
                    dist_tmp = []
                    tmp_max_dis = None
                    c_oil = [item for item in oil]
                    flag = True
                    for i in range(begin_time, end_time):
                        v_a = percentage * plane_v[i]
                        v_b = (1 - percentage) * plane_v[i]
                        v_tmp.append([v_a, v_b])
                        if upper_bounds[box_i_a] < v_a or upper_bounds[box_i_b] <
v_b:
                            flag = False
                            break
                        elif c_oil[box_i_a] - v_a < 0 or c_oil[box_i_b] - v_b < 0:
                            flag = False
                            break
                        else:
                            c_oil[box_i_a] -= v_a
                            c_oil[box_i_b] -= v_b
                            c = calMassCenters(oil, theta=plane_theta[i])
                            c_gt = [0, 0, 0]
                            dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2],

2)) ** 0.5
                            if tmp_max_dis is None:
                                tmp_max_dis = dis
                            else:
                                if dis > tmp_max_dis:
                                    tmp_max_dis = dis
                            dist_tmp.append("%d, %s" % (i, tmp_max_dis))

                    if flag:
                        if best_max_dis is None or (tmp_max_dis is not None and
tmp_max_dis < best_max_dis):
                            best_max_dis = tmp_max_dis
                            best_c_oil = c_oil[:]
                            v_tmp_best = v_tmp
                            good_percentage = percentage
                            dist_best = dist_tmp
                if best_c_oil is not None:
                    oil = best_c_oil[:]
                    max_dis = best_max_dis
```

```
                                v = v_tmp_best
                                good_percentages.append(good_percentage)
                                choice_tmp_dist = dist_best
                    elif len(choice) == 3:
                        if 1 in choice and 2 in choice and 6 not in choice:
                            choice.sort()
                            box_i_a = 1 - 1
                            box_i_b = 2 - 1
                            box_i_c = choice[-1] - 1
                            c_oil = [item for item in oil]
                            best_c_oil = None
                            best_max_dis = None   # percentage 去刷新，取最大值最小的一种
方案
                            v_tmp_best = []
                            good_percentage = None
                            dist_best = []
                            for percentage in percentages:
                                tmp_max_dis = None
                                c_oil = [item for item in oil]
                                flag = True
                                v_tmp = []
                                dist_tmp = []
                                for i in range(begin_time, end_time):
                                    v_b = percentage * plane_v[i]
                                    v_c = (1 - percentage) * plane_v[i]
                                    v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                    v_tmp.append([v_a, v_b, v_c])
                                    if upper_bounds[box_i_b] < v_b or upper_bounds[box_i_c] <
v_c:
                                        flag = False
                                        break
                                    elif c_oil[box_i_b] - v_b < 0 or c_oil[box_i_c] - v_c < 0:
                                        flag = False
                                        break
                                    else:
                                        if c_oil[box_i_a] - v_a < 0:
                                            v_a = 0
                                        c_oil[box_i_b] -= (v_b - v_a)
                                        c_oil[box_i_c] -= v_c
                                        c_oil[box_i_a] -= v_a
                                        c = calMassCenters(oil, theta=plane_theta[i])
                                        c_gt = [0, 0, 0]
                                        dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2],
```

```
2)) ** 0.5
                                    if tmp_max_dis is None or dis > tmp_max_dis:
                                        tmp_max_dis = dis
                                    dist_tmp.append("%d, %s" % (i, tmp_max_dis))
                            if flag:
                                if best_max_dis is None or (tmp_max_dis is not None and
tmp_max_dis < best_max_dis):
                                    best_max_dis = tmp_max_dis
                                    best_c_oil = c_oil[:]
                                    v_tmp_best = v_tmp
                                    good_percentage = percentage
                                    dist_best = dist_tmp
                        if best_c_oil is not None:
                            oil = best_c_oil
                            max_dis = best_max_dis
                            v = v_tmp_best
                            good_percentages.append(good_percentage)
                            choice_tmp_dist = dist_best
                    elif 5 in choice and 6 in choice and 1 not in choice:
                        box_i_a = 6 - 1
                        box_i_b = 5 - 1
                        choice.sort()
                        box_i_c = choice[0] - 1
                        c_oil = [item for item in oil]
                        best_c_oil = None
                        best_max_dis = None  # percentage 去刷新，取最大值最小的一种
方案
                        v_tmp_best = []
                        good_percentage = None
                        dist_best = None
                        for percentage in percentages:
                            tmp_max_dis = None
                            c_oil = [item for item in oil]
                            flag = True
                            v_tmp = []
                            dist_tmp = []
                            for i in range(begin_time, end_time):
                                v_b = percentage * plane_v[i]
                                v_c = (1 - percentage) * plane_v[i]
                                v_a = min(v_b * beiyong_per, upper_bounds[box_i_a])
                                v_tmp.append([v_c, v_b, v_a])
                                if upper_bounds[box_i_b] < v_b or upper_bounds[box_i_c] <
v_c:
```

```
                                    flag = False
                                    break
                                elif c_oil[box_i_b] - v_b < 0 or c_oil[box_i_c] - v_c < 0:
                                    flag = False
                                    break
                                else:
                                    if c_oil[box_i_a] - v_a < 0:
                                        v_a = 0
                                    c_oil[box_i_b] -= (v_b - v_a)
                                    c_oil[box_i_c] -= v_c
                                    c_oil[box_i_a] -= v_a
                                    c = calMassCenters(oil, theta=plane_theta[i])
                                    c_gt = [0, 0, 0]
                                    dis = (pow(c_gt[0] - c[0], 2) + pow(c_gt[1] - c[1], 2) +
pow(c_gt[2] - c[2],

2)) ** 0.5
                                    if tmp_max_dis is None or dis > tmp_max_dis:
                                        tmp_max_dis = dis
                                    dist_tmp.append("%d, %s" % (i, tmp_max_dis))
                            if flag:
                                if best_max_dis is None or (tmp_max_dis is not None and
tmp_max_dis < best_max_dis):
                                    best_max_dis = tmp_max_dis
                                    best_c_oil = c_oil[:]
                                    v_tmp_best = v_tmp
                                    dist_best = dist_tmp
                                    good_percentage = percentage
                        if best_c_oil is not None:
                            oil = best_c_oil
                            max_dis = best_max_dis
                            v = v_tmp_best
                            good_percentages.append(good_percentage)
                            choice_tmp_dist = dist_best



                # oil_map[tuple(choice)] = oil
                if max_dis is None:
                    continue
                if min_dis is None or min_dis > max_dis:
                    min_dis = max_dis
                    final_choice = choice
                    best_oil = [item for item in oil]
```

```
                v_best = v
                best_dist = choice_tmp_dist
        last_oil_total = sum(last_oil)
        last_oil = [item for item in best_oil]
        # pprint.pprint(oil_map)
        # print(final_choice)
        # print(v_best)
        # print(min_dis)
        favor_flag = True
        for l in range(split_time):
            if max([float(best_dist[p].split(',')[1]) for p in range(len(best_dist))]) <= favor:
                favor_flag = False
                l = split_time
                break
            if l < 60:
                if float(best_dist[l].split(',')[1]) > favor:
                    favor_flag = False
                    l = split_time
                    break
            else:
                if float(best_dist[l].split(',')[1]) > favor:
                    begin_time = begin_time + l
                    break
        if not favor_flag:
            if final_choice is None:
                kexin_flag = False
                break
            if final_max_min_dis is None:
                final_max_min_dis = min_dis
            else:
                if min_dis > final_max_min_dis:
                    final_max_min_dis = max(final_max_min_dis, min_dis)
            # for d in best_dist:
                # print(d)
            begin_time += split_time

        else:
            if final_choice is None:
                kexin_flag = False
                break
            if final_max_min_dis is None or min_dis > final_max_min_dis:
                final_max_min_dis = max([float(best_dist[p].split(',')[1]) for p in range(l)])
            # for d in best_dist[:l]:
                # print(d)
```

```
            # print(need_oil)
            # print(last_oil)
            # print(abs(last_oil_total - (need_oil + sum(last_oil))) < 0.01)
            # print()
        print("==========")
        print(split_time)
        if kexin_flag == False:
            print("不行")
        else:
            print("质心距离最大值：%f" % final_max_min_dis)
print(good_percentages)
```